Introduction to XML

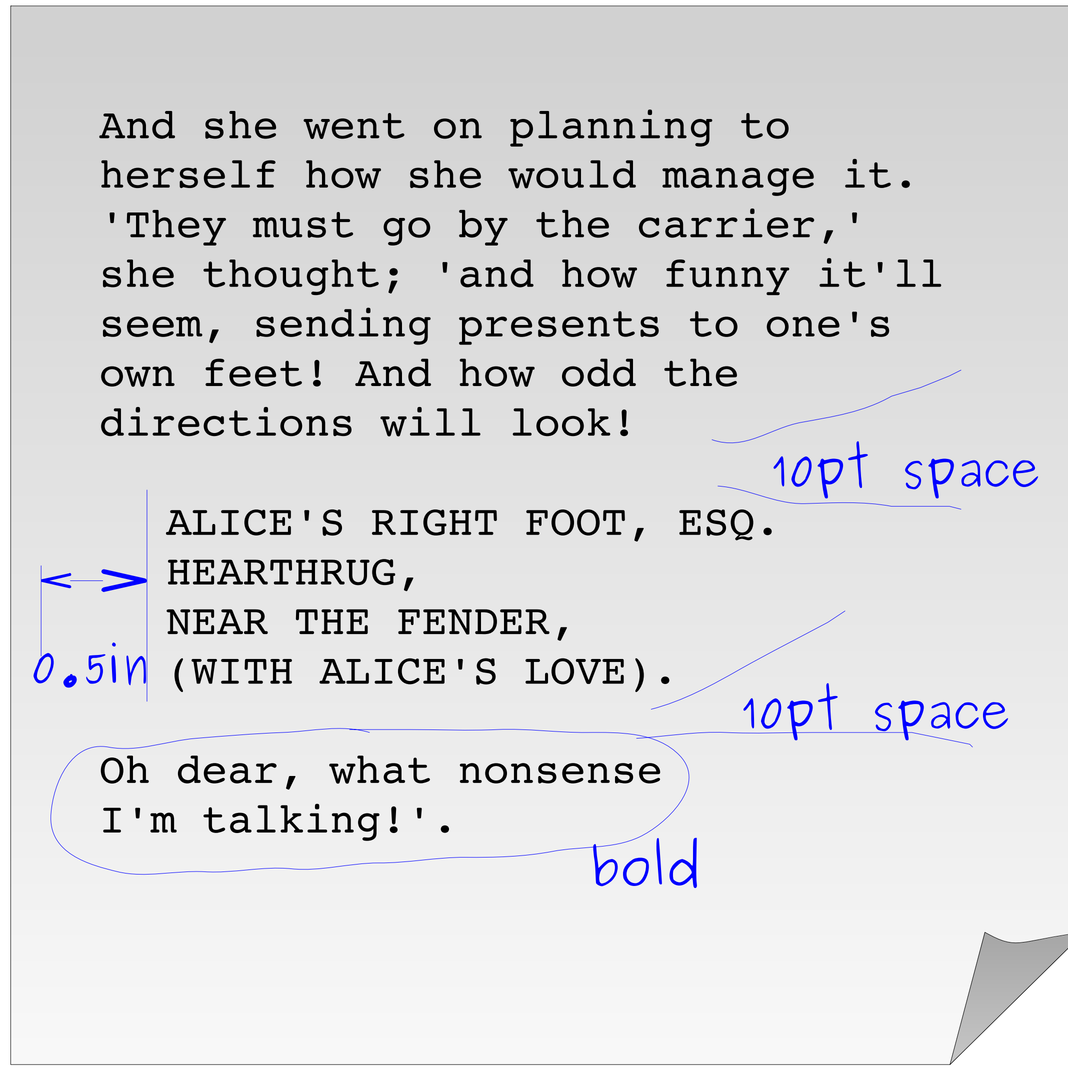
Patryk Czarnik

XML and Applications 2013/2014
Lecture 1 – 7.10.2013

Text markup – roots

The term **markup** origins from hints in manuscript to be printed in press.

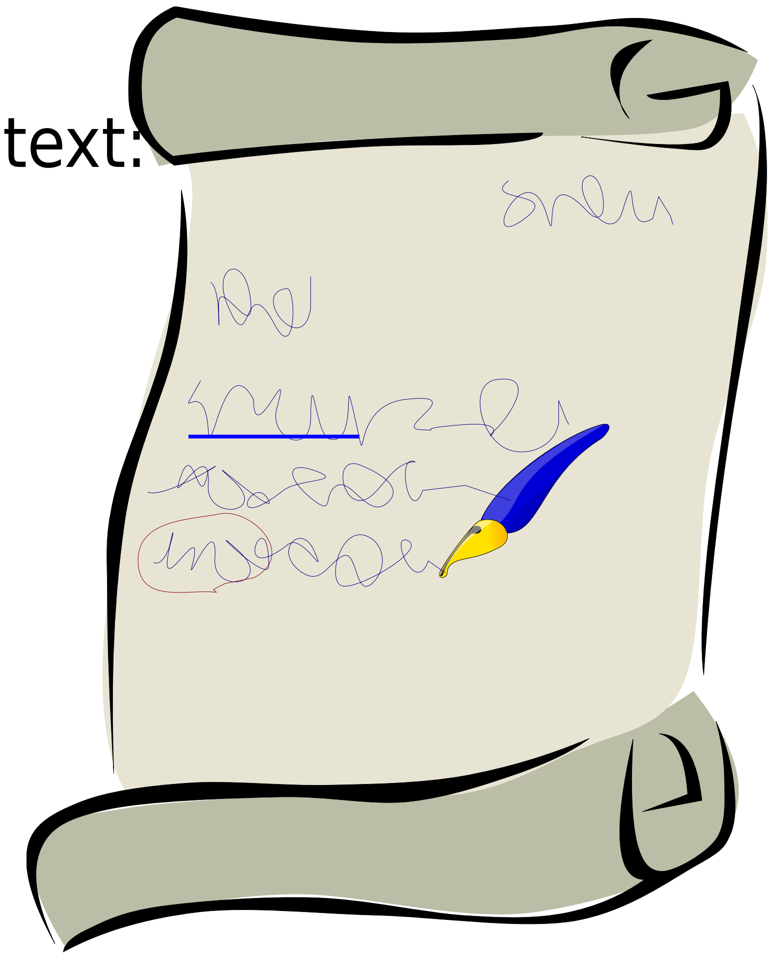
Po polsku
znakowanie tekstu



Text markup – roots

In fact people have marked up text since the beginning of writing.

- Marking up things in hand-written text:
  - punctuation, indentation, spaces,
  - underlines, capital letters.
- Structural documents:
  - layout of letter – implicit meaning,
  - tables, enumeration, lists.
- Today informal markup used in computer-edited plain text:
  - email, forum, blog (FB etc.),
  - SMS, chat, instant messaging.



Text markup – fundamental distinction

Presentational markup

- Describes the appearance of text fragment
  - font, color, indentation,...
- Procedural or structural
- Examples:
  - Postscript, PDF, TeX
  - HTML tags: `<B>` `<BR>`
  - direct formatting in word processors
  - XSL-FO (we will learn)

Semantic markup

- Describes the meaning (role) of a fragment
- Examples:
  - LaTeX (partially)
  - HTML tags: `<STRONG>` `<Q>` `<CITE>` `<VAR>`
  - styles in word processors (if used in that way)
  - most of SGML and XML applications

Documents in information systems

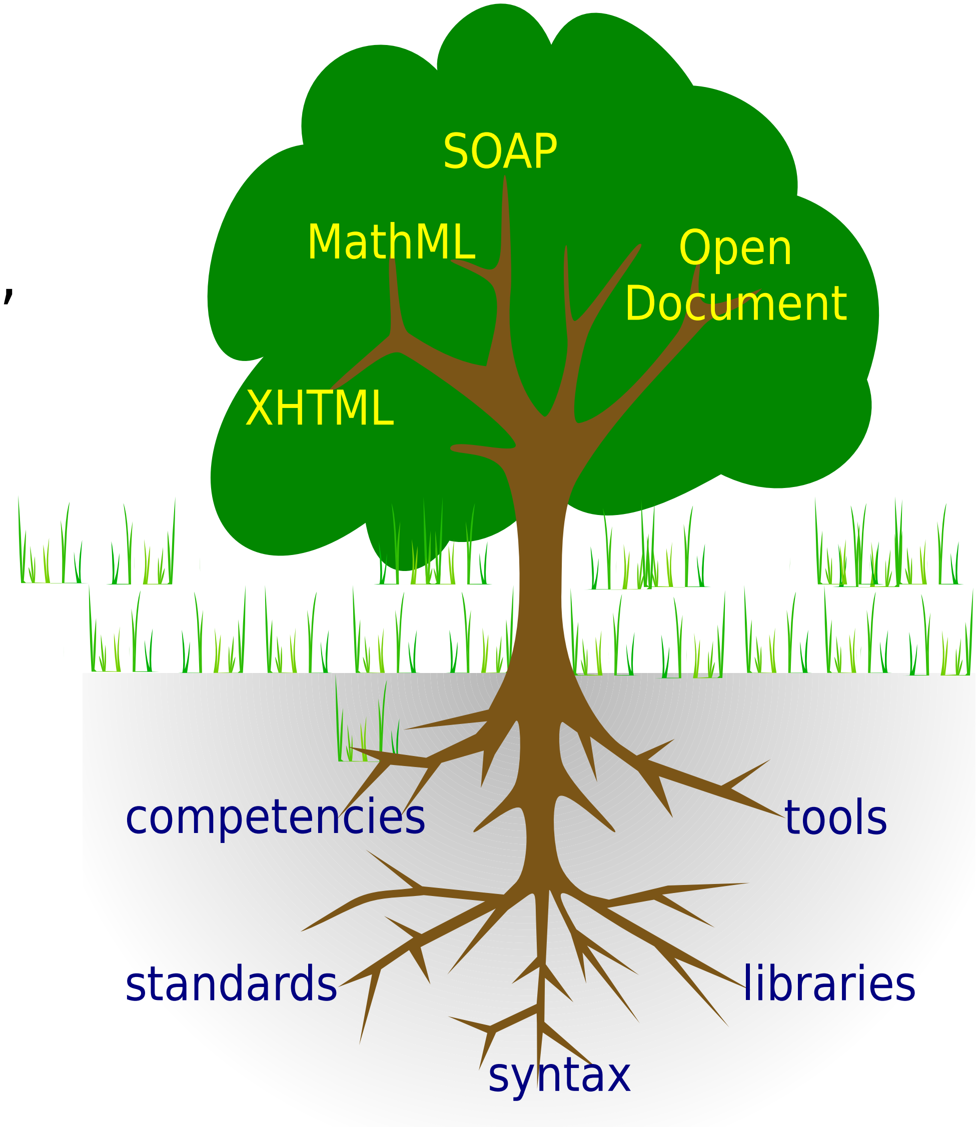
- Since the introduction of computers to administration, companies and homes plenty of digital documents have been written (or generated).
- Serious problem: number of formats, incompatibility.
- De facto standards in some areas (e.g. .doc, .pdf, .tex)
  - most of them proprietary
  - many of them binary and hard to use
  - some of them undocumented and closed for usage without a particular tool

Let's design another format
replacing all existing!

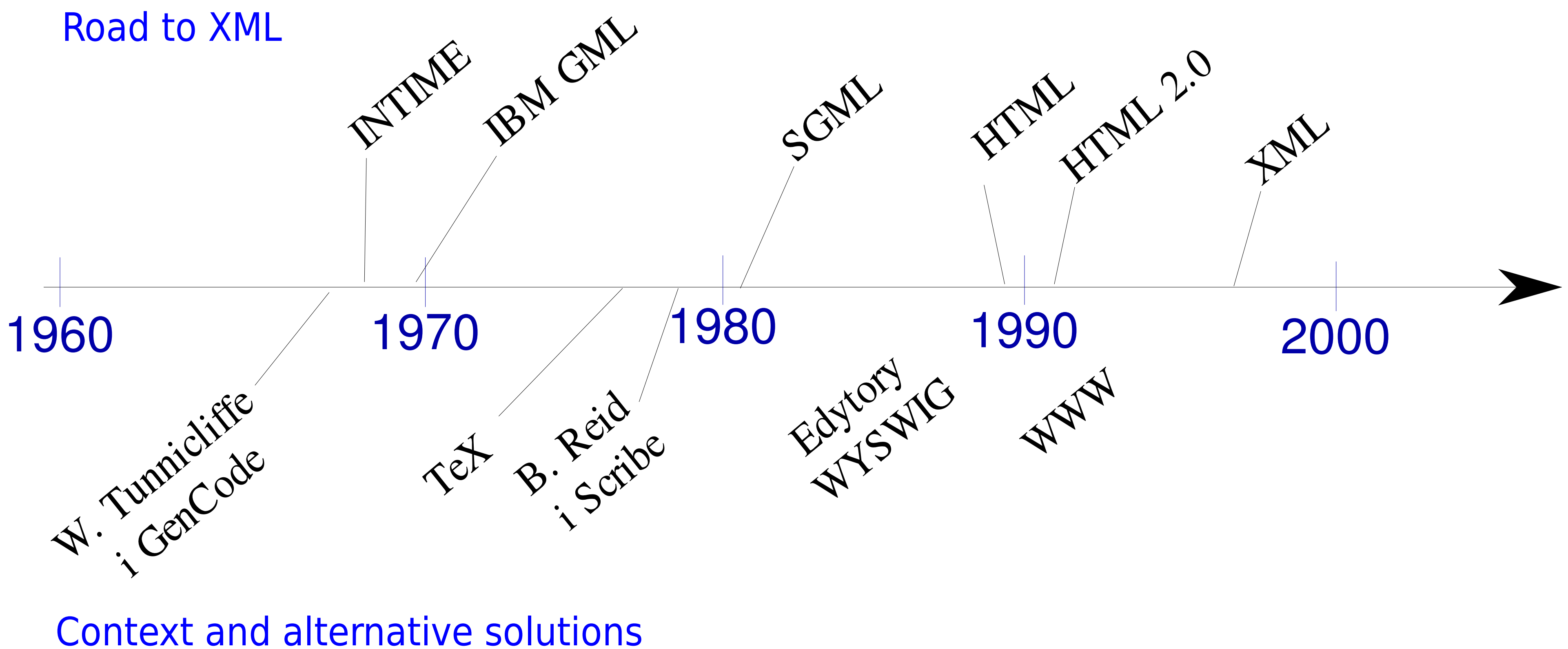
And now we have 1000+1 formats to handle...

Why is XML a different approach?

- Common base
  - document model
  - syntax
  - technical support (parsers, libraries, supporting tools and standards)
- Different applications
  - varying set of tags
  - undetermined semantics
- Base to define formats rather than one format
- General and extensible!



A bit of history – overview



Road to XML

- 1967–1970s – William Tunnicliffe, GenCode
- Late 1960s – IBM – SCRIPT project, INTIME experiment
  - Charles **G**oldfarb, Edward **M**osher, Raymond **L**orie
  - **G**eneralized **M**arkup **L**anguage (GML)
- 1974–1986 – Standard Generalized Markup Language (SGML)
  - ISO 8879:1986
- Late 1990s – E**x**tensible **M**arkup **L**anguage (XML)
  - W3C Recommendation 1998
  - Simplification(!) and subset of SGML

What is XML?

- **Standard** – Extensible Markup Language
  World Wide Web Consortium (W3C) Recommendation
  - version 1.0 – 1998
  - version 1.1 – 2004
- **Language** – a format for writing structural documents in text files
- **Metalanguage** – an extensible and growing family of concrete languages (XHTML, SVG, etc...)
- Means of: (two primary applications)
  - document markup
  - carrying data (for storage or transmission)

What is XML not?

- Programming language
- Extension of HTML
- Means of presentation (for humans)
- Web-only, WebServices-only, database-only,
  nor any other _-only technology – XML is general.
- Golden hammer

XML components
Main logical structure

- **Element** (*element*)
  - start tag (*znacznik otwierający*)
  - end tag (*znacznik zamykający*)
- **Attribute** (*atrybut*)
- Text content / **text node** (*zawartość tekstowa* / *węzeł tekstowy*)

```
<article id="1850" subject="files">
  <author>Jan Kowalski</author>
  <title>File formats</title>
  <p>
    <n>Open document</n> files may have
    the following extensions:
  </p>
  <list type="unordered">
    <item>odt</item>
    <item>ods</item>
    <item>odd</item>
    <item>odp</item>
    <item>odb</item>
  </list>
</article>
```

XML components
Comments and PIs

```
<?xml-stylesheet type="text/css" href="style.css"?>
<article id="1850" subject="files">
  <author>Jan Kowalski</author>
  <?Categorisation technical informal ?>
  <title>File formats</title>
  <!-- <p>Commented content... -->
</article>
<!-- Modified: 2013-10-02T11:11:00 -->
```

- **Comment** (*komentarz*)
- **Processing instruction** (*instrukcja przetwarzania, ew. instrukcja sterująca, dyrektywa*)
  - target (*cel, podmiot*)

XML components – CDATA

- **CDATA section** (*sekcja CDATA*)
  - Whole content treated as a text node, without any processing.
  - Allows to quote whole XML documents (not containing further CDATA sections).

```
<example>
  The same text fragment written in 3 ways:
  <option>x > 0 &amp; x &lt; 100</option>
  <option>x > 0 &#38; x &#60; 100</option>
  <option><![CDATA[x > 0 & x < 100]]></option>
</example>
```

Document prolog

```
<?xml version="1.0" encoding="iso-8859-2" standalone="no"?>
<!DOCTYPE article SYSTEM "article.dtd">
<article>
   ...
</article>
```

- **XML declaration**
  - Looks like a PI, but formally it is not.
  - May be omitted. Default values of properties:
    - `version` = 1.0
    - `encoding` = UTF-8 or UTF-16 (deducted algorithmically)
    - `standalone` = no
- Document type declaration (**DTD**)
  - Optional

Unicode and character encoding

- Unicode – big table assigning characters to numbers.
  - Some characters behave in a special way, e.g.
    U+02DB ˛ Ogonek
- One-byte encodings (ISO-8859, DOS/Windows, etc.)
  - Usually map to Unicode, but not vice-versa
  - Mixing characters from different sets not possible
- Unicode Transformation Formats:
  - UTF-8 – variable-width encoding, one byte for characters 0-127 (consistent with ASCII), 16 bits for most of usable characters, up to 32 bits for the rest
  - UTF-16 – variable-width, although 16 bits used for most usable characters; big-endian or little-endian
  - UTF-32 – fixed-length even for codes > 0xFFFF

XML components
Character & entity references

- **Character reference** decimally: `ü` (*referencja do znaku*)
- Character reference hexadecimally: `ü`
  - Relate to character numbers in Unicode table.
  - Allow to insert any acceptable character even if out of current file encoding or hard to type from keyboard.
  - Not available within element names etc.
- **Entity reference**: `<` `&MyEntity;` (*referencja do encji*)
  - Easy inserting of special characters.
  - Repeating or parametrised content.
  - Inserting content from external file or resource
    - addressable by URL.

Where do entities come from?

- 5 predefined entities: `lt` `gt` `amp` `apos` `quot`
- Custom entities defined in DTD
  - simple (plain text) or complex (with XML elements)
  - internal or external

We skip details of **unparsed entities** and **notations**.

```
<!ELEMENT doc ANY>
<!ENTITY lecture-id "102030">
<!ENTITY title "XML and Applications">
<!ENTITY abstract SYSTEM "abstract.txt">
<!ENTITY lect1 SYSTEM "lecture1.xml">
```

```
<?xml version="1.0"?>
<!DOCTYPE doc SYSTEM "entities.dtd">
<doc>
  <lecture id="&lecture-id;">
    <title>&title;</title>
    <abstract>&abstract;</abstract>
    &lect1;
  </lecture>
</doc>
```

```
<?xml version="1.0"?>
<p>XML is fine.</p>
<p>A general parsed entity is well-formed
if it forms a well-formed XML document
when put between element tags.</p>
In particular, it may contain
text and any number of elements.
```

Document Type Definition

- Defines structure of a class of XML documents (“XML application”).
- Optional and not very popular in new applications.
  - Replaced by XML Schema or alternative standards.
  - It is worth to know it, though. Important for many technologies created 10-30 years ago and still in use.
- Beside document structure definition, which we'll learn in the next week, it allows to define entities and notations.

Associating DTD to XML document (3 options)

- Internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE doc [
  <!ELEMENT doc ANY>
  <!ENTITY title "XML and Apps">
]>
<doc>...
```

- External DTD

```
<?xml version="1.0"?>
<!DOCTYPE doc SYSTEM "entities.dtd">
<doc>...
```

```
<!ELEMENT doc ANY>
<!ENTITY title "XML and Applications">
```

- Mixed approach – internal part processed first and has precedence for some kinds of definitions (including entities)

```
<?xml version="1.0"?>
<!DOCTYPE doc SYSTEM "entities.dtd"
[
  <!ENTITY title "XML and Advanced Applications">
]>
<doc>...
```

External entity identifiers

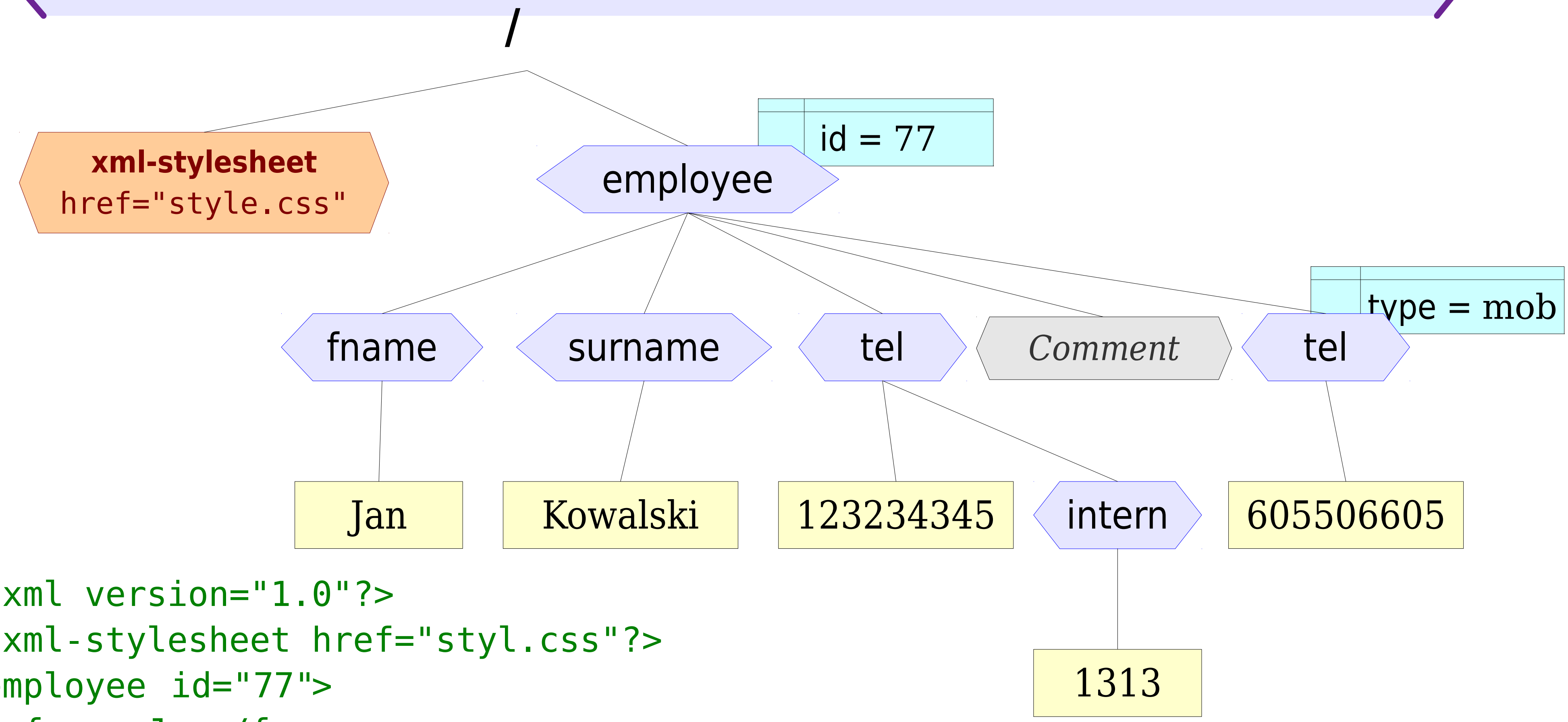
For the external DTD fragment or an entity.

- System identifier
  - `SYSTEM "lecture1.xml"`
  - `SYSTEM "http://xml.mimuw.edu.pl/lecture1.xml"`
- Public identifier
  - `PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"`
  - Public identifiers mapped to actual resources using *catalog file* – SGML-related technology.
  - Some processors (e.g. Web browsers) may use their internal knowledge about a document when they see an expected public identifier.
  - System URI given as additional “fallback” (in XML required, in SGML not).

XML syntax – supplement

- Elements have to be closed (in stack-like order).
  - Shorthand for empty elements: <elem/>
- Two possibilities of attribute value quotation: " or '
- Not every character is allowed in XML document, even by a character reference.
  - Different sets in XML 1.0 and 1.1
- Surprising curiosities:
  - -- is forbidden within comments
  -]]> is forbidden anywhere in text content
    - therefore > is ever needed

Document as a tree




```
<?xml version="1.0"?>
<?xml-stylesheet href="styl.css"?>
<employee id="77">
  <fname>Jan</fname>
  <surname>Kowalski</surname>
  <tel>123234345<intern>1313</intern></tel>
  <!-- Comment -->
  <tel type="mob">605506605</tel>
</employee>
```

Language or metalanguage?

- XML is a language.
  - Grammar, additional constraints expressed descriptively → one can determine whether a sequence of characters is well-formed XML.
- Better to think as of a **metalanguage**.
  - Common base for defining particular languages
  - Set of languages (open, unlimited)

XML application

- **XML application** (*zastosowanie XML*)
  - A concrete language with XML syntax
- Typically defined as:
  - Fixed set of acceptable tag names (elements and attributes, sometimes also entities and notations)
  - Structure enforced on markup, e.g.:
    "`<person>` may contain one or more `<first-name>` and must contain exactly one `<surname>`"
  - Semantics of particular markups (at least informally)
- Means of defining XML applications (the syntax part):
  - DTD – part of XML standard
  - XML Schema – W3C Recommendation
  - Relax NG and other alternative standards

Two levels of document correctness

- Document is **well-formed** (*poprawny składniowo*) if:
  - conforms to XML grammar,
  - and satisfies additional *well-formedness constraints* defined in XML recommendation.
  - Then it is accessible by XML processors (parsers).
- Document is **valid** (*poprawny strukturalnie, "waliduje się"*) if additionally:
  - is consistent with specified document structure definition; from context: DTD, XML Schema, or other;
  - in strict sense (DTD): satisfies *validity constraints* given in the recommendation.
  - Then it is instance of a logical structure and makes sense in a particular context.

Two faces of XML

“Text document”

- Flexible structure, mixed content
- Text (formatted or annotated with tags)
- Content created and used by humans

```
<p><spoken who="alice">Curiouser
  and curiouser!</spoken> cried Alice
  <remark>she was so much surprised, that
  for the moment she quite forgot how to
  speak good English</remark>;
  <spoken who="alice">now I'm opening
  out like the largest telescope.</spoken>
</p>
```

“Database”

- Strict structure
- Various datatypes
- Created and processed automatically

```
<order nr="18/2013">
  <customer id="1313"/>
  <order-date>2013-10-10</order-date>
  <deliv-date>2013-11-03</deliv-date>
  <items>
    <item good-id="56312" qty="1"/>
    <item good-id="56100" qty="10"/>
    <item good-id="56560" qty="7"/>
  </items>
</order>
```

Applications of XML

- Traditional (successor of SGML) – content management
  - Source text markup – preferably semantic – to be used in various ways (publication, searching, analysis)
  - Combining documents (links, references, etc.)
- Modern – data serialisation, programming technologies
  - Saving structural data in files
  - Integration of distributed applications: “web services” (SOAP), REST, AJAX
  - Databases (import/export, “XML databases”)
  - Format of configuration files for many technologies
- Somewhat between – IMO the best place for XML:
  - Structural documents (forms etc.) to be processed by IT systems

XML vs (X)HTML

HTML

- Defined set of elements and attributes
- Their meaning established
- Defined (to some extent) way of presentation
- Although specification exists, tools support (and often create) incorrect HTML.

XML

- All (syntactically correct) tag names allowed
- Undefined semantics
  - <p> is not necessarily a paragraph!
- Unspecified way of presentation
- Processors obliged to work with well-formed XML

XML vs SGML

SGML

- *“Convenient for author”*
- Some ambiguity allowed when supported by DTD,
  - e.g. in HTML `<p>` or `<li>` may stay not closed
- Token attributes allowed to be unquoted
- More datatypes for attributes in DTD
- More DTD structuralisation capabilities
- DTD required

XML

- *“Convenient for processor”*
- Strict unambiguous syntax
- Less options, simpler DTD
- Unified with modern internet standards (URI, Unicode)
- DTD optional

What can we do with XML?

- Define new XML-based formats using XML Schema or other standards
  - Validate documents against the definition
- Edit manually (e.g. Notepad) or using specialised tools
- Store in files or databases, transfer through network
- Process documents (read, use, modify or create, write) in custom applications
  - Use existing parsers and libraries
- Search and query for data using XQuery, XPath, XSLT, or custom applications
- Transform to other formats (for presentation, but not only) using XSLT, XQuery, or custom applications
- Format using stylesheets or specialised tools

Advantages of XML

- Compared to binary formats:
  - Readable (to some extent...) for humans, “self-descriptive”
    - Possibility to read or edit using simplest tools
    - Easier debugging
- Compared to ad-hoc designed formats:
  - Common syntax and document model
    - Common way of defining XML applications (XML Schema)
  - Existing tools, libraries, and supporting standards
  - **Interoperability**
- Compared to WYSIWYG editors and their formats:
  - Semantic markup available, more advanced than *flat* styles
  - Relatively easy conversion to other formats (using transformations and stylesheets)

Drawbacks of XML

- Verbosity
  - Writing numbers, dates, images, etc. as text not efficient
  - Syntax of XML (e.g. element name repeated in closing tag)
  - Common use of whitespace for indentation
    (not obligatory, of course)
- Complexity
  - Inherited features of SGML (entities, notations, even whole DTD) which are rarely used in modern applications, but have to be supported by processors
- Technical restrictions, e.g.:
  - Elements can not overlap (trees, not DAGs)
  - Binary content not allowed (there are some solutions – we will learn)
  - Requirement of exactly one root element impractical

Where does XML make sense?

- Text-oriented applications
  - As source format for further processing
  - To denote metadata, structural dependencies, links, etc.
- Data-oriented applications
  - When structural text or tree-like structure appears in a natural way; e.g. business documents interchange
  - When interoperability more important than efficiency
    - public administration services, external business partners, heterogeneous environment

But XML (read also “WebService”) is maybe not the best format to transfer arrays of numbers between nodes performing a physical process simulation (in a centralised and internal solution).

Alternatives to XML – text

For text-oriented applications of XML:

- TeX and LaTeX
- Direct tagging in graphical text editors
  - flat styles
  - more advanced solutions, e.g. Adobe FrameMaker
- “Lightweight markup”
  - MediaWiki
  - AsciiDoc, OrgMode, and others

```
==== A dialogue ====

"Take some more [[tea]],
" the March Hare said to Alice,
 very earnestly.

"I've had nothing yet,"
Alice replied in an offended tone:
"so I can't take more."

"You mean you can't take ''less'',"
 said the Hatter: "it's '''very'''
easy to take ''more'' than nothing."
```

MediaWiki source example from Wikipedia

Alternatives to XML – data

For “modern” applications of XML:

- JSON (JavaScript Object Notation)
  - more compact than XML
  - often used instead of XML in AJAX-like solutions
- YAML
  - similar to JSON, but more advanced
- CSV
  - simple and poor
- ASN.1, EDIFACT
  - different approach (not as generic)


```
{ "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber": [
    {"type": "home",
     "number": "212 555-1234"
    },
    {"type": "fax",
     "number": "646 555-4567"
    }
  ]
}
```


JSON example from Wikipedia

Namespaces – motivation

- Same names of tags may denote different things.
- Problematic especially when combining document fragments from different sources into one document.

```
<article code="A1250">
  <title>Assignment in Pascal and C</title>
  <author>
    <fname>Jan</fname> <surname>Mądralski</surname>
    <address>...
      <code>01-234<code>
    </address>
  </author>
  <body>
    <paragraph>
      Assignment is written as <code>x = 5</code> in C
      and <code>x := 5</code> in Pascal.
    </paragraph>
  </body>
</article>
```

XML namespaces – realisation

- **Namespace name** (*identyfikator przestrzeni nazw*) – globally unique identifier
  - Universal Resource Identifier (URI) in XML v1.0
  - Internationalized Resource Identifier (IRI) in XML v1.1
- **Namespace prefix** (*prefiks przestrzeni nazw*) – local, for convenient reference
  - Local for document or fragment
  - Processors should not depend on prefixes!
- Names resolved and interpreted as pairs: (namespace name, local name)
- To make things more complex:
  - scope and overrding
  - default namespace

Usage of namespaces and prefixes

```
<art:article code="A1250"
  xmlns:art="http://xml.mimuw.edu.pl/ns/article"
  xmlns:t="http://xml.mimuw.edu.pl/ns/text-document"
  xmlns:ad="urn:addresses">
  <art:title>Assignment in Pascal and C</art:title>
  <art:author>
    <fname>Jan</fname> <surname>Mądralski</surname>
    <ad:address>...
      <ad:code>01-234</ad:code>
    </ad:address>
  </art:author>
  <art:body>
    <t:paragraph>
      Assignment is written as <t:code>x = 5</t:code> in C
      and <t:code>x := 5</t:code> in Pascal.
    </t:paragraph>
  </art:body>
</art:article>
```

Namespaces – overriding and scopes

```
<pre:article code="A1250" xmlns:pre="http://xml.mimuw.edu.pl/ns/article">
  <pre:title>Assignment in Pascal and C</pre:title>
  <pre:author>
    <fname>Jan</fname> <surname>Mądralski</surname>
    <pre:address xmlns:pre="urn:addresses">...
      <pre:code>01-234</pre:code>
    </pre:address>
  </pre:author>
  <pre:body>
    <pre:paragraph xmlns:pre="http://xml.mimuw.edu.pl/ns/text-document">
      Assignment is written as <pre:code>x = 5</pre:code> in C
      and <pre:code>x := 5</pre:code> in Pascal.
    </pre:paragraph>
  </pre:body>
</pre:article>
```

Default namespace

- Applies to element names which do not have a prefix.
- Does not apply to attributes.

```
<article code="A1250" xmlns="http://xml.mimuw.edu.pl/ns/article">
  <title>Assignment in Pascal and C</title>
  <author>
    <fname>Jan</fname> <surname>Mądralski</surname>
    <address xmlns:pre="urn:addresses">...
      <code>01-234</code>
    </address>
  </author>
  <body>
    <paragraph xmlns:pre="http://xml.mimuw.edu.pl/ns/text-document">
      Assignment is written as <code>x = 5</code> in C
      and <code>x := 5</code> in Pascal.
    </paragraph>
  </body>
</article>
```

Namespaces – supplement

- **Qualified name** – name with non-empty ns.URI
- **Unqualified name** – name with null (not assigned) ns.
  - elements without prefixes when no default namespace
  - attributes without prefixes – always
- Namespace name
  - Only identifier, even if in form of an address!
  - Should be in form of URI / IRI; some processors do not check it, though
  - Pay attention to every character (uppercase/lowercase, etc.) – most processors simply compare strings
- XML namespaces may be used not only for element and attribute names – e.g. type names in XML Schema

Namespace awareness

- A document may be well-formed as XML while erroneous from the point of view of namespaces.
  - For some applications (usually old ones...) such document might be proper and usable.
- Modern parsers can be configured to process or not namespaces.

  The mentioned document would be
  - parsed successfully by a parser which is not namespace-aware,
  - revoked by a namespace-aware parser.