XML design

XML – meta language

- Each XML document has both a logical and a physical structure.
- Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity.
- Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup.
- The logical and physical structures must nest properly
- User decides
  - Element names
  - Element order and hierarchy
- DTD or Schema = grammar

DTD

<!– Document type description (DTD) example (part) -->

<!ELEMENT university (department+)>
<!ELEMENT department (name, address)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>

- Document type description, structural description
- one rule /element
  - name
  - content
- a grammar for document instances
- ”regular clauses"
- (not necessary)

DTD use

- validating parser
  - checks that the document conforms to the DTD
- logical use of tags
- existing DTD standards for many application areas
  - common vocabulary
- Replaced by XML Schema in new applications

XML design: Granularity 1

```
<country>
   <state>
    Washington,  Seattle
   </state>
   <state>
    Washington D.C., Washington
   </state>
</country>
```

Granularity 2

```
<country>
<state>
   <state_name>Washington</state_name>
   <capital>Seattle</capital>
</state>
<state>
   <state_name>Washington D.C.</state_name>
   <capital>Washington</capital>
</state>
</country>
```

- The more structure the more tags
- fine or coarse
- fine granularity adds information and allows exact search and fine-tuned formatting

Design issue: data vs structure

- See examples:
  - http://users.metropolia.fi/~jaanah/ElDocCP/material/Pract2bmenu1.xml
  - http://users.metropolia.fi/~jaanah/ElDocCP/material/pract2bmenushort.xml

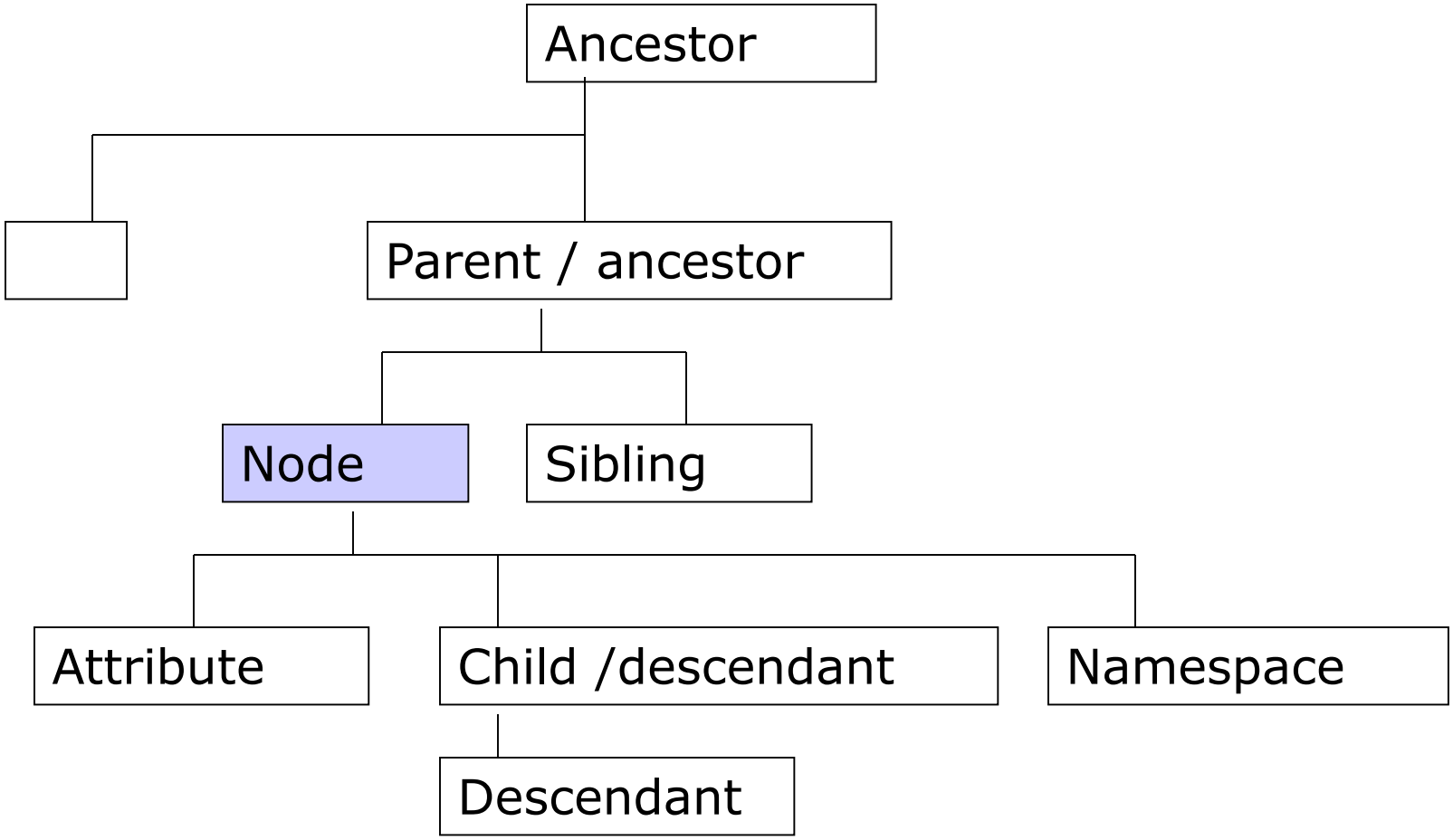
Separate data and structure

```
<cafe>
<dishes>
        <dish type=”main” kind=”seafood”>
                <name>Seafoodplatter</name>
                <price>28</price>
        </dish>
        <dish type=”starter” kind=”vegetable”>
                <name>Redbeet salad</name>
                <price>8,50</price>
        </dish>
…
```

Tree terminology

- document as a tree of nodes
- different types of nodes, including element nodes, attribute nodes and text nodes
- Root element, nodes, leaves
- axis
- child node; descendant
- parent, ancestor
- following-sibling, preceding-sibling

Document tree



Attributes

- Element property or contents
- attached to opening tags (or empty element tags)
  - attribute name
  - attribute value
- only one value
- the value can contain any characters

```
<book author=”Oscar Wilde">
...
</book>

<book keywords="XML SGML">
...
</book>
```

Reserved attributes

- xml:space
  - showing 'white space' or not
  - values: preserve or default
- white space characters:

character	Unicode value
tab	#x9
newline	#xA
carriage return	#xD
space	#x20

- normalization removes extra white space

Reserved attributes

xml:lang

- document language
- ISO 639 (+ ISO 3166, countries)
- or user defined or IANA

```
<product>
<paragraph xml:lang="en">
...
</paragraph>
<paragraph xml:lang="fi">
...
</paragraph>
</product>
```

xml:lang

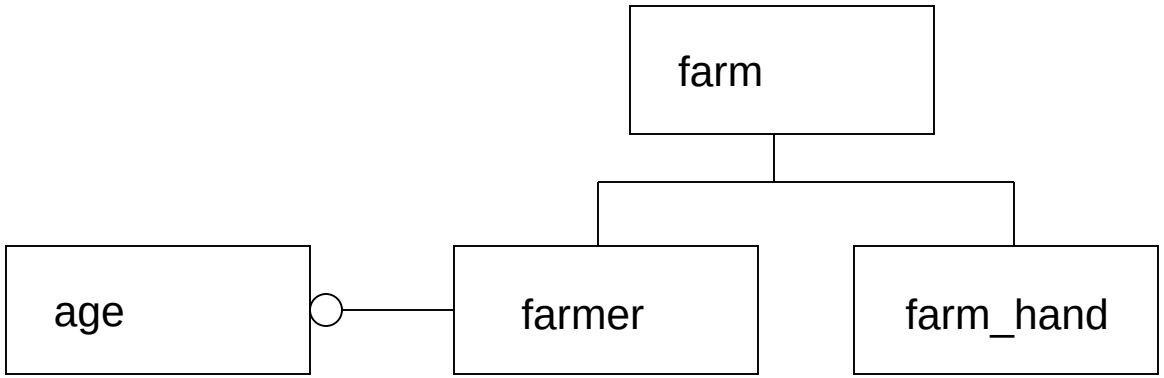
```
<p xml:lang="en">The quick brown fox
jumps over the lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>

<sp who="Faust" desc='leise' xml:lang="de">
<l>Habe nun, ach! Philosophie,</l>
<l>durchaus studiert mit heißem Bemüh'n.</l>
</sp>
```

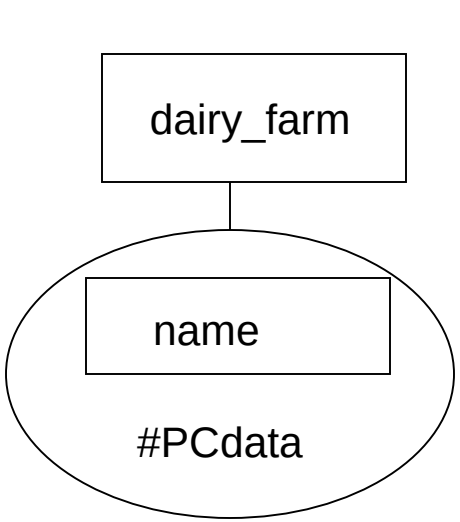
Here are some of the problems using attributes vs. child elements:

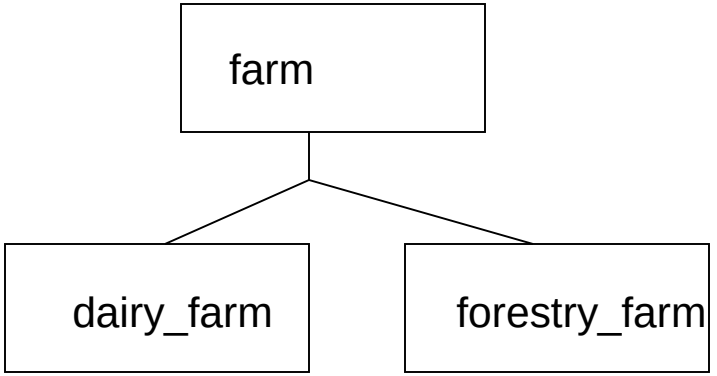
- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable (for future changes)
- attributes cannot describe structures (child elements can)
- attributes may be more difficult to manipulate by program code
- attribute values are not easy to test against a DTD

Tree diagrams



<!ELEMENT farm (farmer, farm_hand)>





<!ELEMENT farm (dairy_farm | forestry_farm)>

SGML

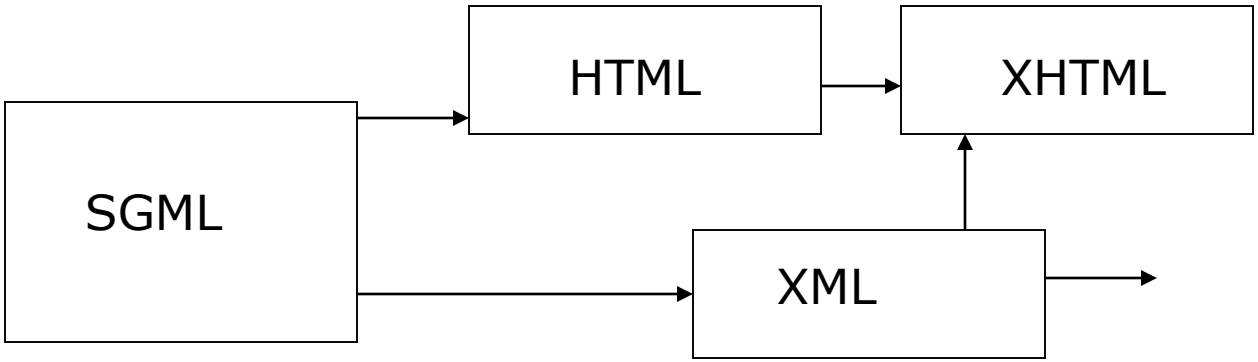
- Standard Generalised Markup Language
- standard 1986 (ISO)
- contains also DTD and style sheets
- complex
- tools are complicated and difficult to develop, they are expensive and not too many
- XML is a subset of SGML
- is typically used in large applications, for example aircraft documentation and paper machine manuals

HTML

- Hypertext Markup Language (and HTTP protocol)
- based on SGML
- a great success
- non-standard extensions
- numerous tools
- predefined format
  - applicable when one needs to show information
  - looks neat on browsers

XML - SGML - HTML

- XML combined features from SGML and HTML
- many tools
- XHTML follows XML recommendation
- cannot solve all problems alone
- all three languages are needed (XML, HTML, SGML)



The design goals for XML 1

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.

The design goals for XML II

6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance

Entities

- Units of storage, for example the document entity: contains the entire document
- general entities
  - can include any well-formed content
- parameter entities (in DTDs)
- internal entities
  - character references
- external entities
  - parsed entities - text
  - unparsed entities - images, sound, binary files

Use of parsed entities

When we have the entity **&metrop;** containing a string of characters
”Helsinki Metropolia University of Applied Sciences"

we could write

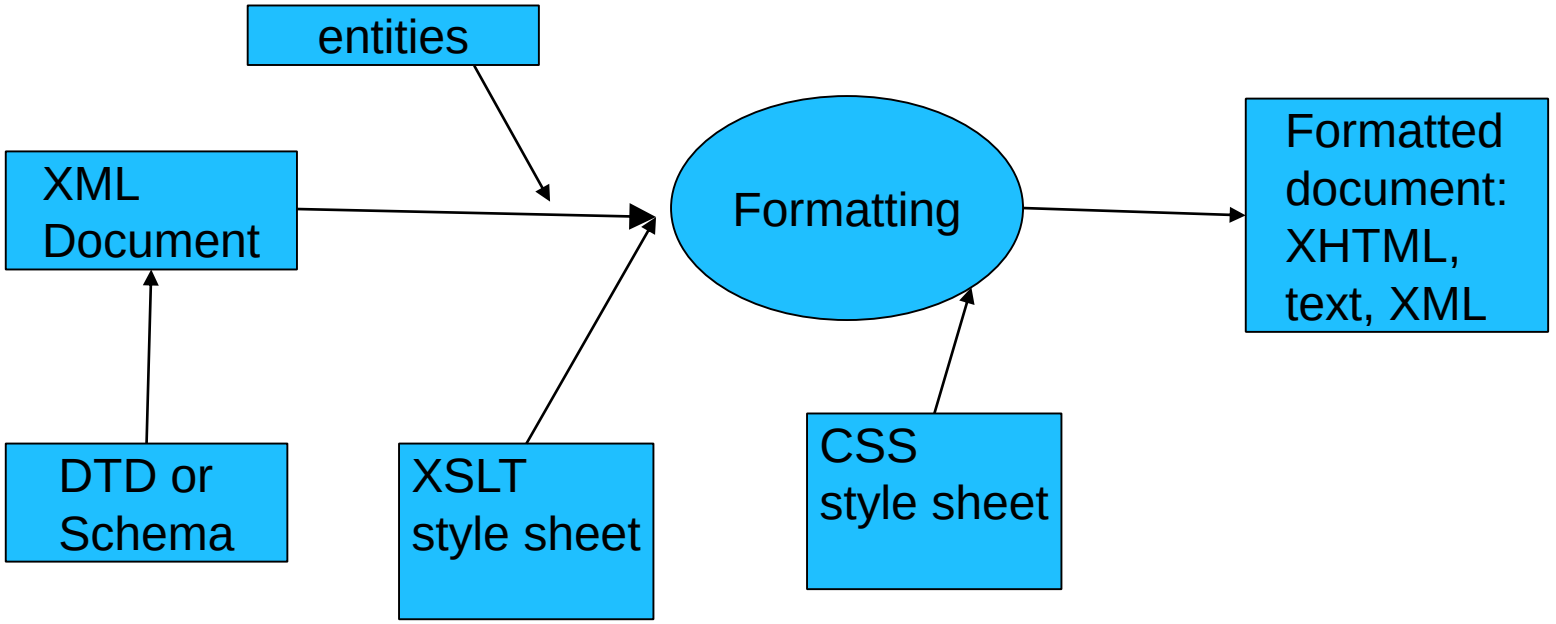
I study at the &metrop;.

- Avoid < and >
- reserved entities: < and >
- some processors are able to interpret
  - if user writes in a math expression < and > processor would interpret as strings < and >

Processing of entities

- XML processors must interpret entities correctly
  - replacement of text (well-formed XML)
  - binary entities are included in the document
  - no recursion is allowed (reference to itself)

Processing of XML documents



XML is the basis for RDF and the Semantic Web

- Resource Description Framework (RDF) is an XML text format that supports resource description and metadata applications
- RDF integrates applications and agents into one Semantic Web
- Formal descriptions of terms in a certain area (shopping or manufacturing, for example) are called ontologies
- for example Dublin Core and RosettaNet

Well-formed documents

- An XML document is well-formed if it
  - has a properly nested (hierarchical) tree structure,
  - complies with the basic syntax and structural rules of the XML 1.0 specification,
  - and its every parsed entity is well-formed

XML parser or processor

- stops when an error is encountered, is not allowed to guess as opposed to HTML browsers
- IE contains MSXML
- all browsers include a parser
- Apache project Xerces (Java, C++)
- Saxon
- .NET Visual Studio