Structuring the Web: an XML briefing

Anthony Finkelstein
University College London
Department of Computer Science

a.finkelstein@cs.ucl.ac.uk
http://www.cs.ucl.ac.uk/staff/A.Finkelstein
Also: http://www.xlinkit.com

UCL

Why Such a Boring Topic?

- Well …
  - Because I think it might actually be useful
- Particularly if you:
  - Manage large datasets
  - Exchange these datasets with others
  - Are concerned with visualisation of this data
  - Write (simple) programs to manipulate data
  - Manage complex structured texts
  - Are a significant web or computer user

    In other words most of UCL!

[And I am doing exciting research in the area]

Outline

- From HTML to XML
- From well-formedness to validity: the DTD or schema
- Parsing XML documents: DOM and SAX
- Transforming XML documents: XSLT
- Related technologies: XLink and XPath

From HTML to XML

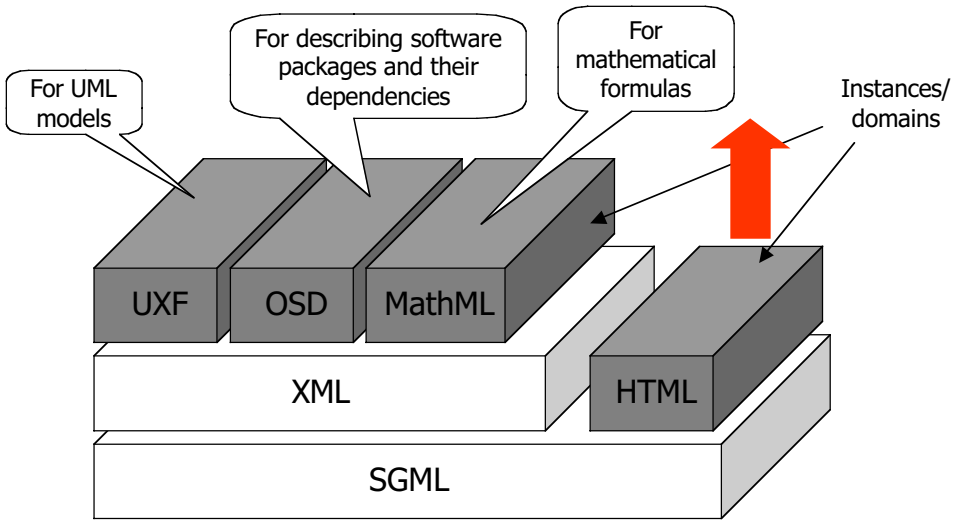
Some of HTML's drawbacks

- HTML isn't extensible
- HTML mixes structure and presentation
  e.g. <TITLE> (structure) and <EM> (presentation)
- HTML isn't reusable
  see above
- HTML has little or no semantic structure
  that's why it's so hard to perform Internet searches

From HTML to XML

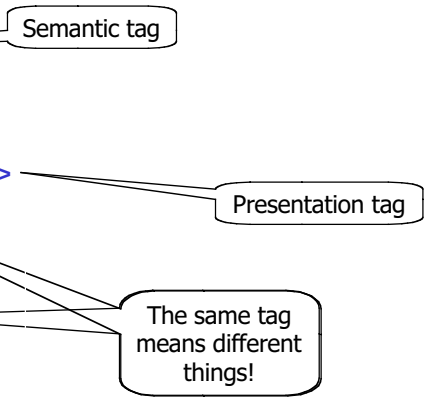
- Is SGML the answer?
  - SGML stands for Standard Generalized Markup Language
  - SGML is a "monster"
    - too complex
    - too complete
    - too extensible
- So why XML?
  - XML stands in the middle
  - XML is a simplification of SGML for general Web use
  - Note: both HTML and XML are *applications* of SGML

SGML, XML, and HTML



HTML

```
<HTML>
<HEAD><TITLE>
Apple Pie Recipe
</TITLE></HEAD>
<BODY>
<H3>My Grandma's Apple Pie</H3>
<H4>Ingredients</H4>
200 g white sugar
4 eggs
…
<H4>Instructions</H4>
Beat eggs and sugar
…
</BODY>
</HTML>
```



HTML

- In the example above:
  - How can a *program* create a shopping list for the recipe's ingredients?
  - The same header <H4> denotes both ingredients and instructions
  - HTML is good for humans, *not* for programs!

XML

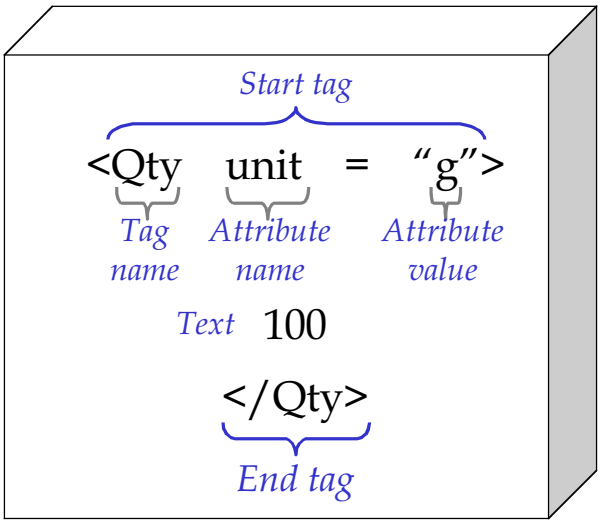
```
<?xml version="1.0"?>
<Recipe>
    <Name>Apple Pie Recipe</Name>
    <Ingredients>
        <Ingredient>
            <Qty unit="g">100</Qty>
            <Item>sugar</Item>
        </Ingredient>
        <Ingredient>
            <Qty unit="units">4</Qty>
            <Item>egg</Item>
        </Ingredient>
    </Ingredients>
    <Instructions>
        <Step>
            Beat eggs and sugar
        </Step>
    </Instructions>
</Recipe>
```

All tags are semantic!

There are no presentation tags!

Some Basic Nomenclature

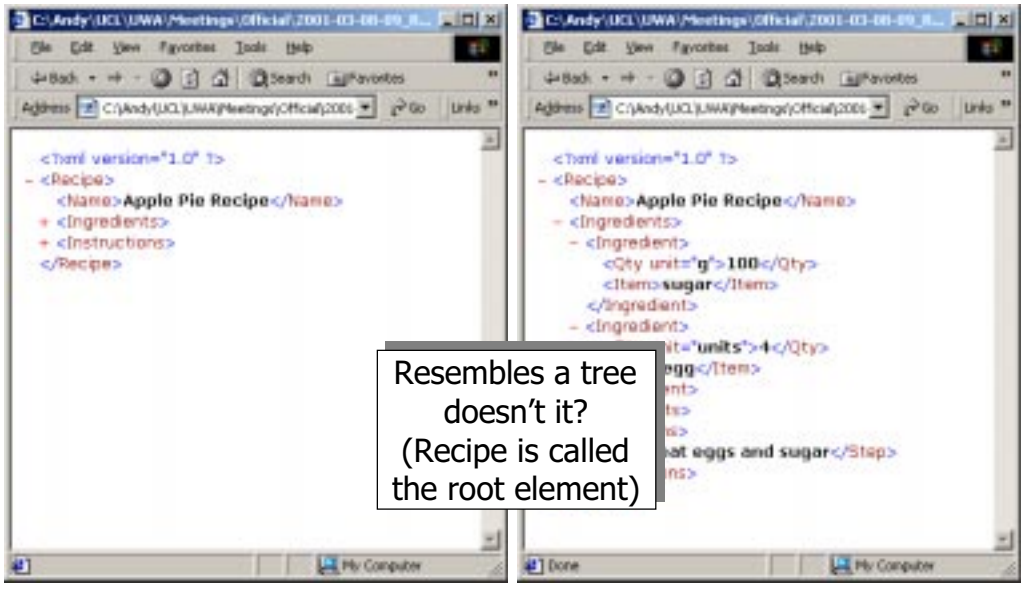
<Qty unit="g">100</Qty>



Well-Formedness

- Unlike HTML, *any* XML document must be well-formed
- Being well-formed means respecting certain rules:
  - No unclosed tags: `<Item> … </Item>`
  - No overlapping tags:
    `<Ingredients> <Ingredient> … </Ingredient> </Ingredients>`
  - Attribute values must be enclosed in quotes
  - The text characters **<**, **>**, and **"** must always be represented by 'character entities'
    In other words, these are *keywords* of the language
    Character entities: < > "
- Well-formed means *parsable*

What Does It All Look Like?



Is Well-Formedness Enough?

- Well-formed means respecting the above rules…

…but…

- …does a well-formed document always make sense?

```
<?xml version="1.0"?>
<Person>
   <FirstName>Andrea</FirstName>
   <LastName>Savigni</LastName>
   <Age>32</Age>
   <Age>30</Age>
   <Age>25</Age>
</Person>
```

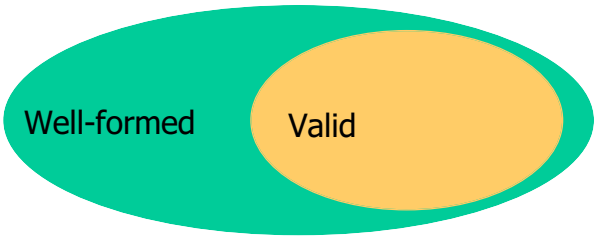
The DTD

or schema

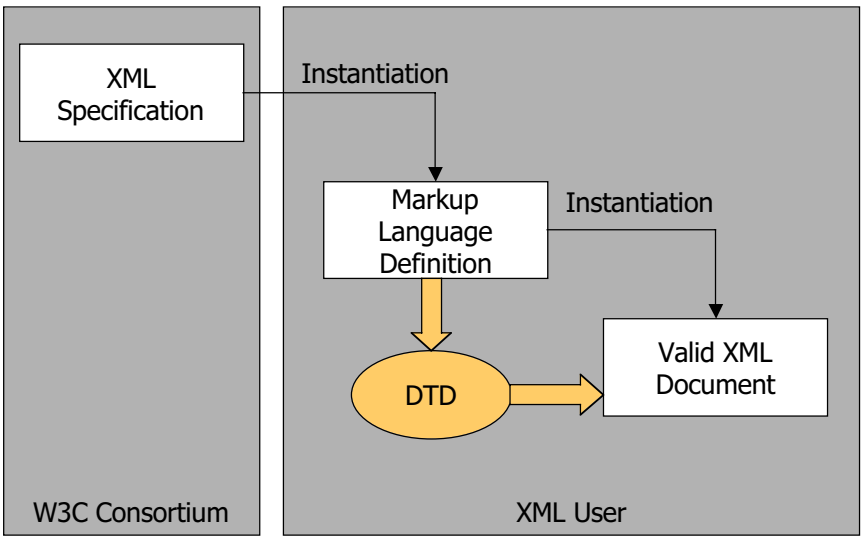
- What it takes is a *grammar* i.e., a set of rules for using and combining tags
  - Such a grammar is called a DTD (Document Type Definition)
  - A DTD actually defines a *new markup language* (sometimes called an "XML dialect")
  - A DTD defines *what tags* are legal, what *attributes* a tag has, how tags are *nested*, how they are *combined*, etc.
  - Countless DTDs have been or are being defined for music, chemistry, mathematics, …

Validity

- A valid XML document is:
  - a well-formed XML document
  - that, *in addition*, conforms to a DTD

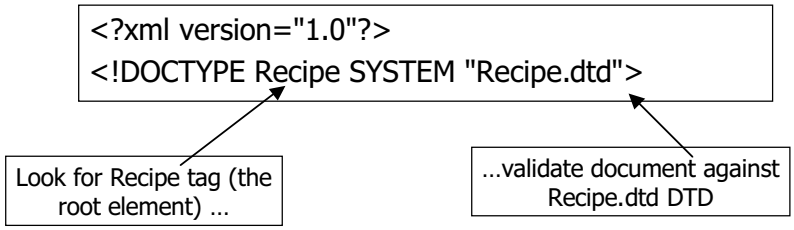


Instantiation



Parsing Valid XML Files

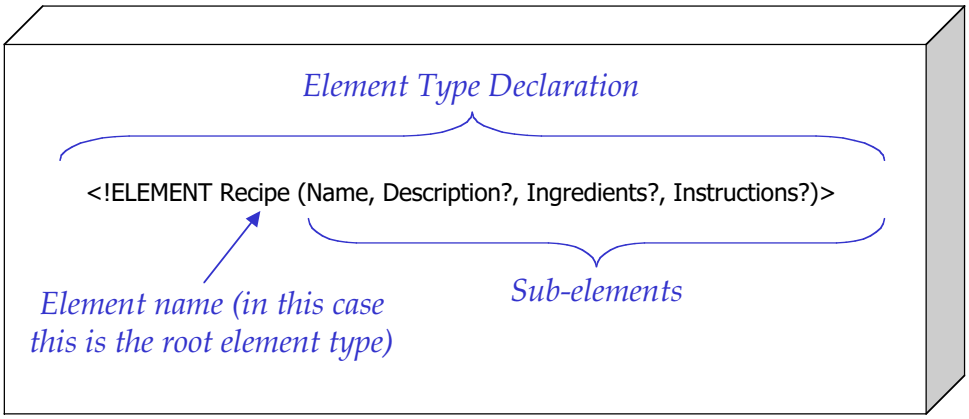
- *Nonvalidating parsers* just check for well-formedness
- Validating parsers check for well-formedness and *then* check for validity
- A valid XML file must contain a *document type declaration* by which the validating parser can retrieve the DTD



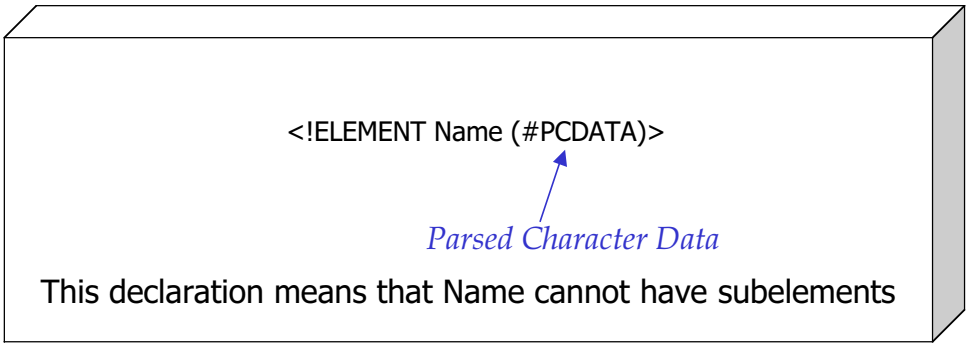
The `Recipe` DTD

```
<!-- This is an example DTD for the recipe markup language -->
<!ELEMENT Recipe (Name, Description?, Ingredients?, Instructions?)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Ingredients (Ingredient)*>
<!ELEMENT Ingredient (Qty, Item)>
<!ELEMENT Qty (#PCDATA)>
<!ATTLIST Qty unit CDATA #REQUIRED>
<!ELEMENT Item (#PCDATA)>
<!ATTLIST Item optional CDATA "0" isVegetarian CDATA "true">
<!ELEMENT Instructions (Step)+>
<!ELEMENT Step (#PCDATA)>
```

An Overview of The DTD Syntax

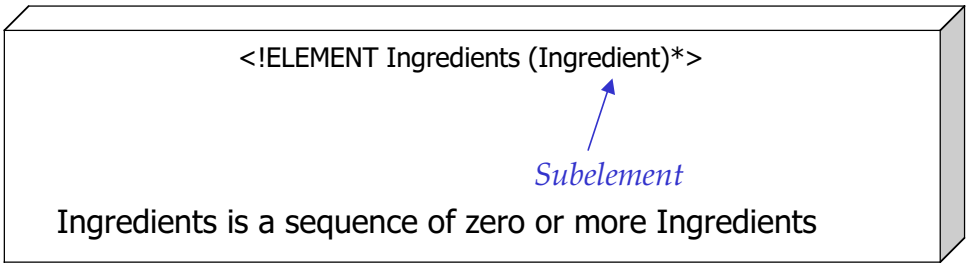


An Overview of The DTD Syntax



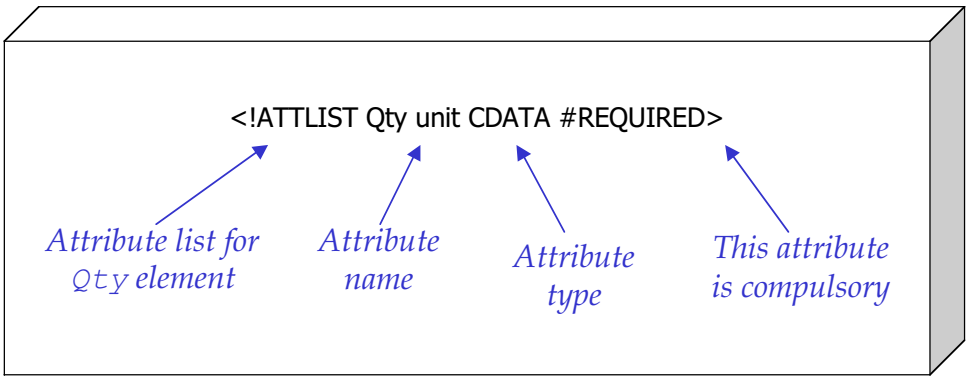
This declaration means that Name cannot have subelements

An Overview of The DTD Syntax



- (Element)? Optional: zero or one occurence of Element
- (Element)* means zero or more occurences of Element
- (Element)+ means one or more occurences of Element

An Overview of The DTD Syntax



XML Virtues So Far

- XML is extensible
  - anyone can create their own markup language
- XML is independent of rendition
  - in other words, an XML document is pure content
- XML is easy to parse
  - plenty of commercial and public-domain parsers are available
  - they can parse *any* well-formed XML document
  - no need to build custom parsers!

The DOM (Document Object Model)

- The DOM is a *specification*
  - i.e., a reference model (like ISO/OSI or TCP/IP)
  - that, of course, can be implemented
  - a DOM implementation is an API
  - language abuse: the DOM is an API (you can use it as long as you remember it's an abuse)
- DOM *bindings* exist for a number of programming languages
- The DOM allows you to manipulate an XML document as a *tree of objects*

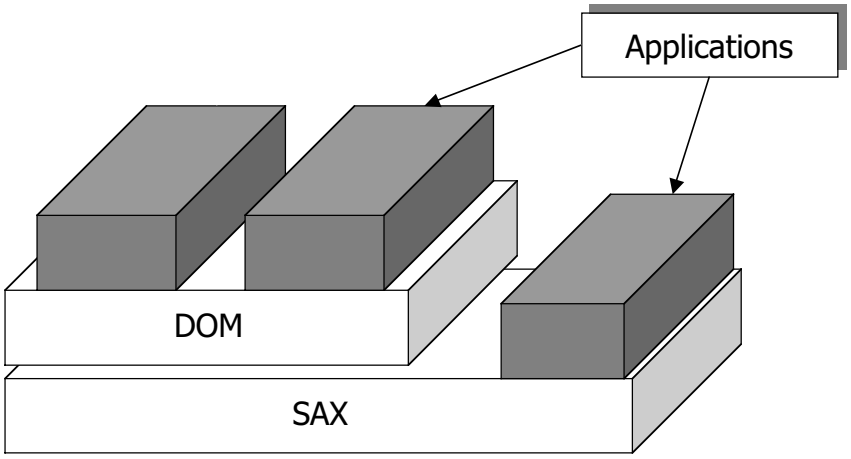
Pros and Cons of the DOM

- Pros:
  - the DOM is extremely simple and easy to use: one method call to process an entire document
  - the DOM is very high-level
  - the DOM is readily available for a number of languages
- Cons:
  - too coarse-grained (it always slurps the whole file, cannot process a part of it)
  - what if the file is huge?
  - what if the file is on a remote machine?
  - keeps the whole tree in memory

The SAX (Simple API for XML)

- SAX is an *event-based* parsing API
  - the parser reads the XML document once
  - at each parser event it notifies the application
  - callback-style mechanism: applications must register appropriate event handlers
- SAX is lower-level than DOM
  - actually, DOM *uses* SAX
  - harder to use but more flexible and efficient

SAX and DOM



What is XSLT?

- XSLT is a language for *transforming* the structure of an XML document
- Hence the legitimate question: Why do I need to *transform* an XML document into another one?
  - communication with another computer: OK, everybody uses XML, but not everybody uses the same DTDs
  - presentation (i.e., communication with a human): the same XML file can be transformed into HTML, WML, PDF, RTF, ...

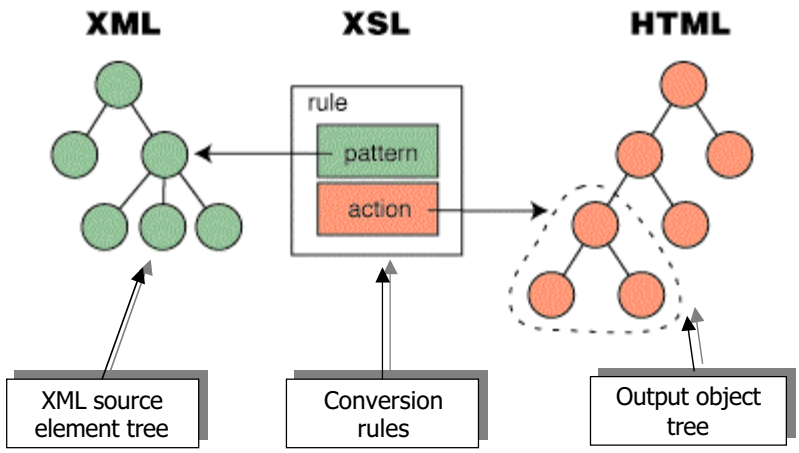
Where Does XSLT Fit?

- XSLT is part of a larger language, called XSL (eXtensible Stylesheet Language)
- XSL covers *formatting* and *presentation* of XML documents
- It soon became clear that this is a two-stage process:
  - transformation (e.g., reordering, sorting, adding a table of contents, etc.), covered by XSLT (eXtensible Stylesheet Language: Transformations)
  - actual rendition, covered by XSL-FO (XSLT (eXtensible Stylesheet Language: Formatting Objects), not yet standardised

Why Is a Separate Language Needed?

- SAX and DOM allow to quickly and easily manipulate XML documents, so why a dedicated language for transforming?
- Once again the answer is: *convenience*
  - XSLT is a *declarative* language
  - provides a huge set of very high-level constructs
  - a very appropriate analogy: SQL
- XSLT converts a document tree into another one without the need to specify the exact sequence of actions to perform

The Classical Example: XML to HTML



Some Key Features of XSLT

- XSLT is declarative
  - no need to specify the sequence of operations
  - even though space is left for scripts (just like in SQL)
- XSLT is written in XML itself!
- XSLT has no side-effects
- Processing is described as a set of independent pattern-matching rules

The Recipe Example: the XML Source

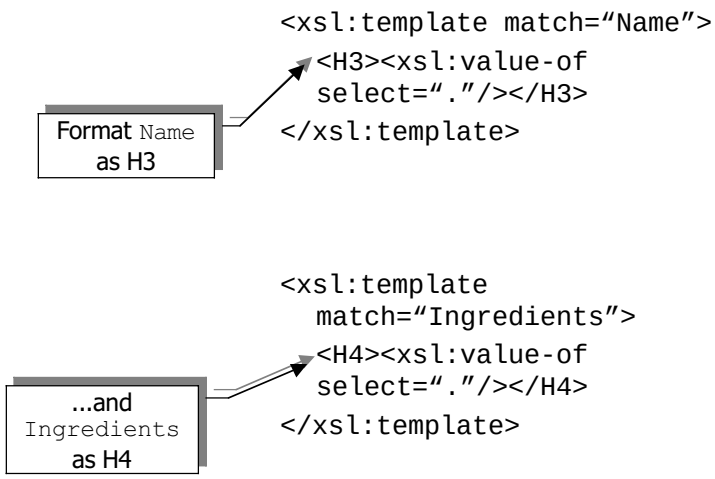
```
<?xml version="1.0"?>
<Recipe>
    <Name>Apple Pie Recipe</Name>
    <Ingredients>
        <Ingredient>
            <Qty unit="g">100</Qty>
            <Item>sugar</Item>
        </Ingredient>
        <Ingredient>
            <Qty unit="units">4</Qty>
            <Item>egg</Item>
        </Ingredient>
    </Ingredients>
    <Instructions>
        <Step>
            Beat eggs and sugar
        </Step>
    </Instructions>
</Recipe>
```

The Recipe Example: the HTML Target

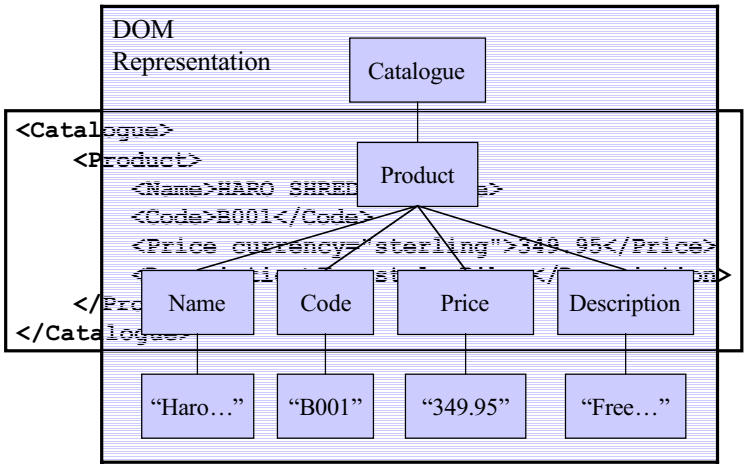
```
<HTML>
<HEAD><TITLE>
  Apple Pie Recipe
</TITLE></HEAD>
<BODY>
<H3>My Grandma's Apple Pie</H3>
<H4>Ingredients</H4>
200 g white sugar
4 eggs
...
<H4>Instructions</H4>
Beat eggs and sugar
...
</BODY>
</HTML>
```

The Recipe Example: the XSLT Rules

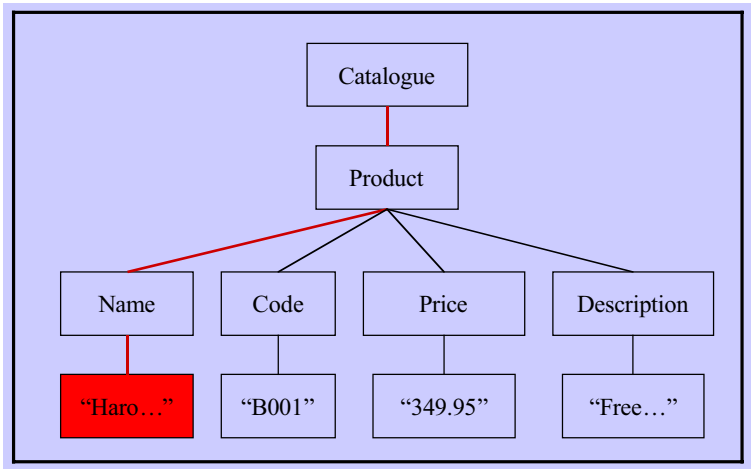
The Recipe Example: the XSLT Rules (cont.)



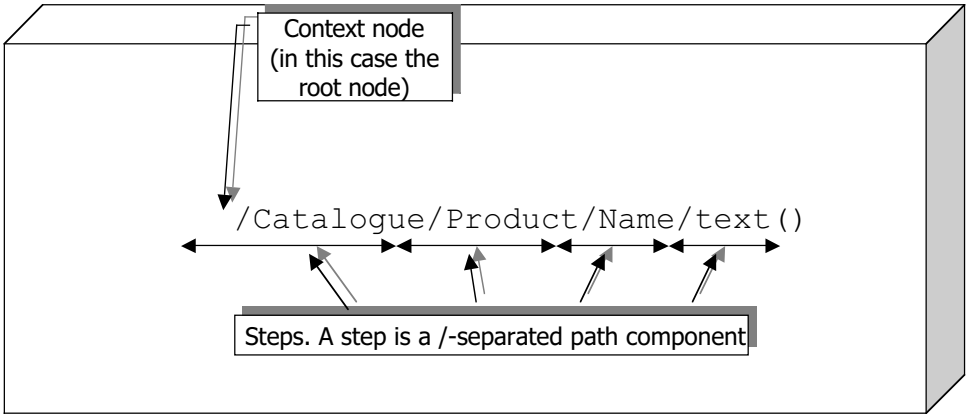
A Quick Recap: DOM Representation



XPath



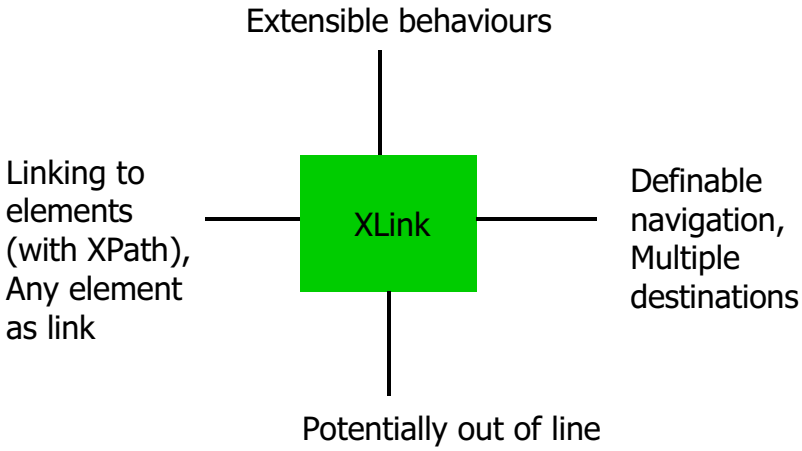
XPath Expressions



Axes

- XPath is *very* powerful!
- Provides many different ways of traversing the tree (the *axes*)
  - the *descendant axis* (`//`) can cover any number of nodes
  - the *parent axis* (`..`)
  - the *attribute axis* (`@`) steps into the attribute nodes of an element

XLink



XLink

- Linking means declaring a relationship between two things
- In HTML: <A HREF="...">
  - the source end of the link knows it is a link
  - the target end:
    - does *not* know (if it's a whole page)
    - *does* know (if it's a part of a page – anchors)
- The basic XLink idea: neither end should know about the link
  - the link resides with a third part

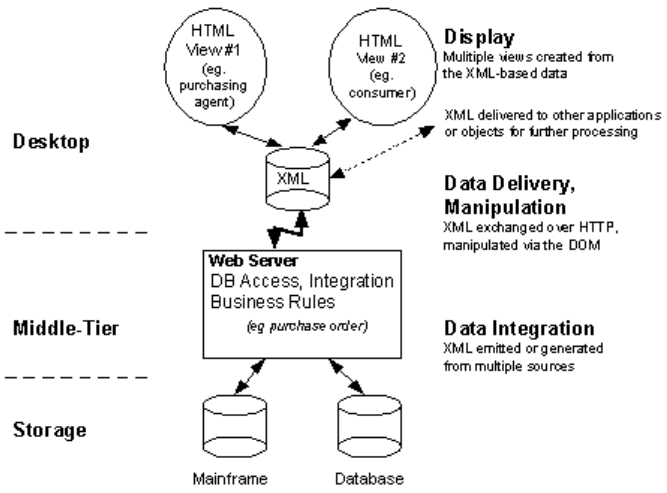
Simple links

- Much like HTML links:

```
<citation xlink:type="simple"
          xlink:href="http://genius.at.work.com">
Savigni(2020)
</citation>
```

- What makes this a link is *not* its name!
  - you can call it whatever you want
- It is the xlink:type attribute

XML Application



XML Application

- database interchange:
  - example: home health care in the US (data interchange between hospitals and health agencies)
    - current: log into hospital, see records in browser, print them and key them into own database
    - XML: log into hospital, drag records onto own database
  - present different web views to clients
  - tailored information discovery

XML Application

- distributed processing:
  - scheduling applications: airlines, trains, buses, subways, restaurants, movies, plays, concerts, …
  - commercial applications: shopping
  - educational applications: online help
  - customer-support applications: lawn-mower maintenance to support for computers
- view selection: switch between views without downloading data again
  - dynamic TOC without data reload
  - switching between languages
  - sorting phone books

XML Applications

- web agents:
  - intelligent searches over the web

    search criteria and searched documents have to be expressed in standard format (e.g. XML); structural requirements beyond HTML;
  - example: 500-channel cable TV and personalised TV guide across entire spectrum of providers

    user preferences and program description in XML

Conclusion

- Statement of Belief:
  - These technologies are already having a major effect on information management! This is only set to increase.
- Questions
  - What are the implications for your research?
  - What XML dialects are relevant to your area?
  - What stake do you have in the relevant standardisation processes?
- And my research - just check out http://www.xlinkit.com