

### 3. Representation of Information – Document Mark-up and XML

“Document Markup” is the term referring to the notion of using a standard format to create and publish documents which would allow them to be exchanged and manipulated more easily. First attempts go back to 1960’s which eventually led to Standard Generalised Mark-Up Language (SGML), an ISO standard since 1986. This uses a logical mark-up in terms of a logical structure and content, instead of a physical layout which is about how a document looks. This yields documents which are independent of platform, system, vendor and version ... truly appealing. But ... SGML documents are very powerful but contain some complex features and this has been their problem.

SGML allows the creation of document vocabularies, so it isn’t really an actual mark-up language itself, it’s a language to allow the specification of other languages, in which documents can be marked-up. This is important.

What happened with SGML is that it was great, but it was/is too great. With SGML you can do really powerful things like define a markup language for a large complex inter-linked document like online documentation, or a diplomatic edition and transcription of an Old Irish manuscript (cf. ISOS project) and doing this difficult task is a really difficult thing for specialist publishers only, but if you want to do something simple, like do up a memo, letter or web page, then doing this in SGML is almost as difficult. It has been termed using a sledgehammer to crack open a nut !

The web and HTML especially can be regarded as an experiment that got out of the lab too soon ... HTML is a very simple way to describe things, too simple in fact. Separately, the part of the world which was wrestling with implementing metadata realised that SGML could have solved the metadata problem but was too complex ... SGML was great at solving big and complex problems but could not solve little problems at the same time so the W3C started the development of XML in Summer 1996. The motivation for this was poor web searching and the idea is that the publishers of information also include metadata descriptions of that information, or perhaps the publishers of information (you, me, DCU, Amazon, Ireland.com, everybody ...) produce that information ONLY in metadata format, and browsers or display devices render it as appropriate. When everybody does that then searching will be much easier and we will have much higher precision (no junk) ... putting the onus on the content providers to index their content properly instead of hoping the search engines can identify the content.

So what is HTML ... the *de facto* standard for publishing web content, and HTML is actually an SGML vocabulary (that means that HTML can be defined by SGML). HTML is not the only markup language around ... there is RTF, EDIFACT, LaTeX, as well as XML.

HTML was/is successful because :

- It is simple and easy to learn with only a few dozen tags, only some of which are used regularly ... NOTE: ... why would one have to actually learn HTML ... shouldn't it be hidden from users ?
- HTML is almost human-readable.
- Hypertext linking which is included in HTML, across unrelated systems does work well. Publicly accessible remote documents can be accessed easily by embedding URLs in HTML just as easy as local documents.
- HTML is portable and HTML documents can be viewed on any platform.
- HTML browsers were/are free and with plug-ins can be come even more powerful.
- HTML document-browsers were easy to build into existing applications like MSWord, Visual Developer Studio, etc.

So, HTML is a success and allowed accelerated adoption of the internet by a large range of users and also allowed easy publishing of information by a large range of users.

But ... the web has pushed HTML far beyond what it was designed for and some of the HTML limitations are:

- HTML provides no mechanism to allow mark-up of something according to its semantics, because the tagset is closed.
- HTML has very little internal structure ... it is almost too easy to write a "valid" HTML document even though the internal semantics of the mark-up may not make sense – for example, we can embed a H1 within a H2, which does not make sense ... HTML is almost too forgiving and there is not really a concept of "valid" HTML since browsers will even display invalid HTML without complaint.
- Because of the limited functionality of HTML, web site developers, especially ecommerce, are writing HTML documents that function as applications thus clogging the Internet with unnecessary client-server traffic and using their desktop PCs as mere screens.
- The HTML tagset was chosen to be manipulated by humans, but not by machines.
- HTML linking capabilities are basic 1-to-1, and require anchors.
- Browser developers have introduced browser-specific tags, a situation which seems resolved now, but for a while it was chaotic.
- Browser developers have introduced tags which are specific to physical layout – e.g. Font. When these are used they hard-encode some semantic aspect, in a presentation tag.
- As a presentation tool, HTML is weak lacking many fundamental page formatting such as hanging indents, control of white space, justification, kerning, etc.

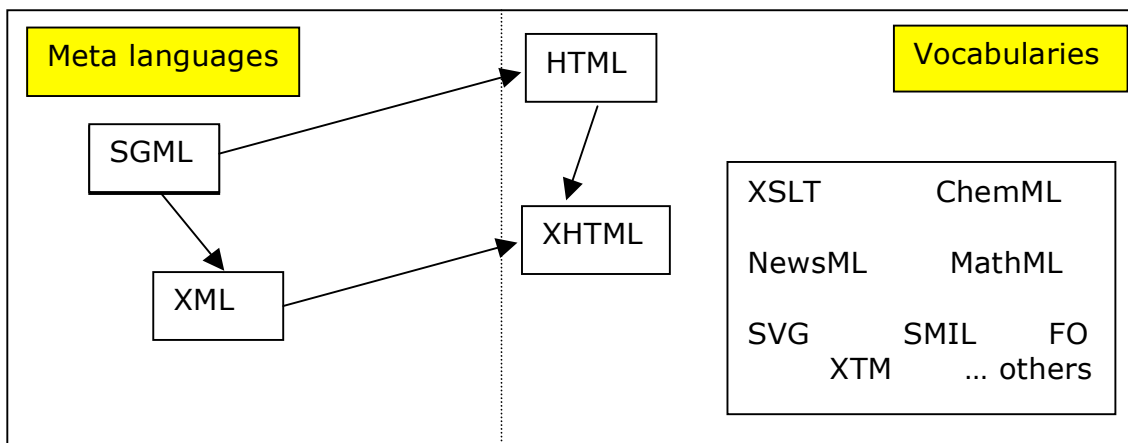
Since the release of the first HTML specification there have been many updates ... HTML, HTML+, HTML 2, HTML Netscape Extensions, HTML 3.2, HTML 4.0 and the flurry of refinements and extensions seems to have died down, though not because the problems have been solved. The industry seems to have moved from extending HTML to inventing XML.

XML is ...

- EXTensible Mark-up Language, a simplified subset of SGML.
- Can be used to define document mark-up vocabularies (XHTML)
- XML documents have a strictly defined structure which is encoded in a DTD and which allows a user-definable hierarchical document structure.
- Retains the powerful features of SGML, namely an extensible language, encoding document structure and allows validation of documents.
- Easier to use and implement than SGML because it ignores the complexity of SGML.
- XML documents look similar to HTML documents but they separate the structure from the presentation.
- A metalanguage, used to describe other languages

XML is used in B2B communications, Electronic Publishing, information access, anywhere there is a need for content exchange, i.e. potentially everywhere.

How do SGML, HTML, XML and others fit together ?



XML uses tags, just like HTML but one can invent one's own tags. A browser won't know what to do with the tags but a search engine might. HTML is a wonderful vehicle for displaying content quickly and it is reasonably portable and the formatting power is adequate but the tags are doing formatting and interaction and hypertext all at once. In XML one says what the tags are and one uses a stylesheet to turn the tags into formatting instructions, and one uses the tags in other B2B communications to give a much higher quality content exchange.

XML is extensible so inventing tags is easy. Suppose I have a document ...

CA414: Multimedia Information Systems Course Notes  
Alan Smeaton  
1999-2000

... can be formatted in HTML in many ways to make it pretty but search engines retrieve this when querying "Information on Alan Smeaton", or "Multimedia Notes".

Marked up in raw XML this looks like ...

```
<?xml version='1.0' encoding='ISO-8859-1' standalone='yes'?>
<doc type="notecover" xml:lang="en">
<title><modcode>CA414</modcode> : <modtitle>Multimedia Information
Systems Course Notes</modtitle></title>
<author> <firstname>Alan</firstname> <surname>Smeaton</surname>
</author>
<year_runs> <year>1999</year>- <year>2000</year> </year_runs>
</doc>
```

... and if I do this for all course note front pages I can then develop a stylesheet for this and render/print/display it whatever way I want, different views for different users in different situations. "notecover" is the DTD file, notecover.dtd

... I've just rolled my own XML, and it looks like HTML !

The XML 1.0 specification describes the syntax for XML documents (elements and attributes) and DTDs.

An XML document is an hierarchical data structure using self-definable tags but it is not easy to get a clear grasp on how it all fits together as there are a many other technologies related to XML.

*Note ... in the notes for this course I give an overview of the syntactic conventions of XML and related technologies, but this is not exhaustive ... there are things I leave out because the purpose of these notes is to give the overview while the details can be found in some of the online references. In fact this whole part of the course may seem disorganised and fragmented but that is because the world of XML is evolving very rapidly, and evolving now – there is no stability, which makes it difficult to teach, and to learn !*

**Well-formed XML** must have the following

- An XML declaration
- At least 1 element
- Exactly 1 root element
- A correct nesting of elements
- All elements must have closing tags `<xx></xx>` whether they are empty or not and the tags must have the same case
- Attribute values must be quoted

This defines what well-formed XML documents are but this is not the same as **valid XML**. Well-formed refers to an XML document being syntactically correct.

A **Document Type Declaration** can be internal/embedded or can be an external DTD, i.e. in another outside file. It appears directly after the XML declaration.

A DTD defines the structure or model of XML documents ... it defines the elements and their cardinality, attributes and their default values, entities and notation aggregations. It is stored in a plain text file and used to **validate** an XML document.

```
<!ENTITY % part "(title?, (paragraph | section)*)">
```

```
<!ELEMENT doc (title, author+, chapter+, appendix*)>  
<!ATTLIST doc type (book | article) "book"  
            isbn CDATA #REQUIRED>
```

```
<!ELEMENT title (#PCDATA)>  
<!ELEMENT author (#PCDATA)>  
<!ELEMENT chapter %part;>  
<!ELEMENT appendix %part;>  
<!ELEMENT section %part;>  
<!ELEMENT paragraph (#PCDATA | url | ol)*>  
<!ATTLIST paragraph type CDATA #IMPLIED.
```

```
<!ELEMENT ol (item+)>  
<!ELEMENT item (paragraph+)>  
<!ELEMENT url (#PCDATA)>
```

... comma means "followed by", | means logical OR, ? means zero or one occurrences, + means one or more, \* means zero or more, #PCDATA means parsable character data, i.e. simple text.

... a part has an optional title and any number of paragraphs and sections where a chapter, appendix and section are defined as parts. A document has a title, 1 or more authors, 1 or more chapters, and 0 or more appendices. A document also has a type, whether a book or article and if it is a book then an ISBN is required. Title, author are both character strings ... and so on.

This defines the structure of any number of instances of XML documents which draw from this DTD. Applications may require all documents to be consistent instances of a particular vocabulary and the DTD indicates what structures and names can be used in a document. Documents are then constructed and named in a conformant manner and documents can be validated in order to find inconsistencies.

**A Valid XML document** is XML that is **well-formed** and that in addition, conforms to a DTD in that all elements and attributes in the document are declared within a DTD, either an internal or an external one.

In order to allow web content providers to move to XML – while allowing backward compatibility with older browsers and allowing content developers to take advantage of XML tools, **XHTML** is a reformulation of HTML 4 as an XML vocabulary. This generates well-formed HTML which won't tolerate the kind of HTML errors browsers presently tolerate.

An example XHTML document ...

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
  <title>A Guide to XML</title>
</head>
<body isbn="12345678">
<h1>Norman Walsh : A Guide to XML</h1>
<h2>Introduction</h2>
<p>The text of this document is ... </p>
<ol>
<li>list items go here</li>
<li>And more go here</li>
</ol>
</body>
</html>
```

## **XHTML vs HTML**

- Documents must be well-formed in XHTML in terms of nesting, closing tags, which is not the case for HTML
- Elements and attribute names must be in lowercase in XHTML, and not so in HTML
- For non-empty elements in XHTML, closing tags are required, and attribute values must always be quoted
- There is handling of white space in attribute values
- Script and style elements are #PCDATA

In a word ... XHTML is a way to prepare for XML from HTML by having manners put on you ;-)

## **Software for Creating and Editing XML Documents**

... apart from applications which produce XML documents directly themselves ... there are dedicated XML editors such as XMetal from SoftQuad.

Other dedicated applications store their documents in XML directly such as SUNs StarOffice suite, CorelDraw, and MS Office 2000 (sort-of !) and MS-Office 2007 definitely (\*.pptx and \*.docx). These are examples of software which uses XML but for which the DTDs are pre-defined and embedded.

Also, many content management systems for managing web content, store their (text) information as XML in an XML document store, and transform that content from XML to HTML on presentation. Makes for easy consistency checking (linking) and easy updating, and easy re-design of websites.

So, that's material on what XML documents look like on the inside, and how to create them, but once they are done, what can one do with them and how can one look at them ?

## **Transforming and Formatting XML Documents**

Cascading Style Sheets (CSS) are used for formatting Web documents. The principle is to select elements by name and apply formatting properties. The CSS syntax is similar to Javascript ...

```
//by name [e.g. <body></body>]
body {
    color : black;
    font-size : 10pt;
    font-family : Verdana, Arian, Helvetica; }
```

```
//by name and class [e.g. <h1 class="chapter"></h1>]
h1.chapter {
    color : red;
    font-weight : bolder;
    text-decoration : underline ; }
```

CSS is still the best style sheet language supported by mainstream browsers who will support HTML and also XML with CSS. External and internal references to CSS are possible within XML documents and they are easy to maintain and to configure which is a nice way to get a consistent look and feel for a set of web pages.

```
<html>
<head>
  <link rel="stylesheet" href="style.css" type="text/css">
  <style type="text/css"
    H1 {text-indent : 10pt}
  </style>
</head>

<body>
  <p style="text-indent : 10pt">Indented Paragraph appears here</p>
</body>
</html>
```

Example of an external style sheet

Example of an internal style sheet

Example of an inline style

Style sheets are associated with an XML document using the *xml-stylesheet* processing instruction. This instruction should be placed after the XML declaration but before the first element

```
<?xml version = "1.0" ?>
<?xml-stylesheet href="doc.xsl" type="text/xsl" ?>
<?xml-stylesheet href="doc.css" type="text/css" alternate="yes" ?>
<doc>
...
</doc>
```

So, what is **XSL** ?

The eXtensible Stylesheet Language (XSL) consists of a language for transforming XML documents (just like CSS) but also a language for specifying formatting properties, and it is specifically for XML documents. It uses an XML syntax and XSL is a combination of :

- XML Path Language (XPath)
- XML Transformations (XSLT)
- XSL (Formatting Objects)

**Why bother with XSL ?** Because it separates the content of a document from its presentational information. An XML document by definition has no associated formatting properties and CSS was developed for HTML which does have a default formatting for HTML elements like H1, H2, etc. However, a style sheet should be more powerful than to allow a transformation into HTML, so XSL provides a mechanism for high-quality formatting and layout of XML documents, both online and in print.

XSL is quite complex ... really ! .. possibly even more complex than the other XML technologies, so lets treat each of the 3 components in turn.

## **XPATH**

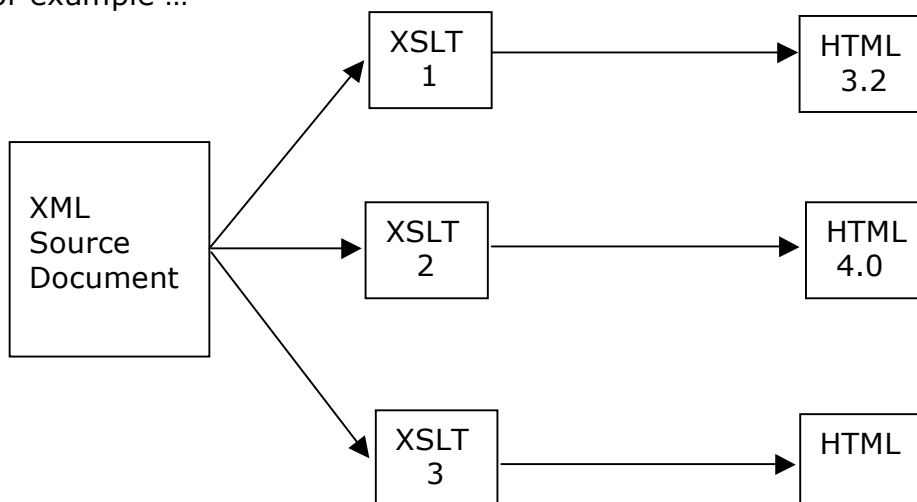
XPath addresses part of an XML document, i.e. it is used to scope or identify parts of an XML document. It is an expression language which allows wildcards and provides the basic facilities for manipulation of strings, numbers and Boolean. XPath is a powerful part of XSL in that it allows us to select a set of ELEMENTs within an XML document based on conditions and based on position within the document hierarchy. It is used to select parts of the document on which we then apply XSL style sheet rules, so in a sense it allows us to “query” XML documents

## **XSL Transformations**

XSL Transformations is a language for transforming XML documents into other XML documents (not necessarily XML to XML but XML to some other XML vocabulary ... like HTML !

It uses a collection of templates or rules to transform the source document where XPath has been used to select the part of a source document to be transformed.

For example ...



In the case above, several XSLT instances can be used to transform the original XML document, depending on the device being used. Some devices may wish to filter out certain fields because of screen size, etc. In such cases the information about what device a user is using (browser version, iPhone, PDA) comes to the web server, and a servlet on the server will pull in the appropriate XSLT to generate the format requested. So, we can use server-side or client-side execution to do the formatting. This is a much more scalable alternative to hardcoding in different HTML versions of a single source of information for different devices.

This can also be used to implement security and filtering of XML fields etc., so it can be quite powerful, much more powerful than CSS.

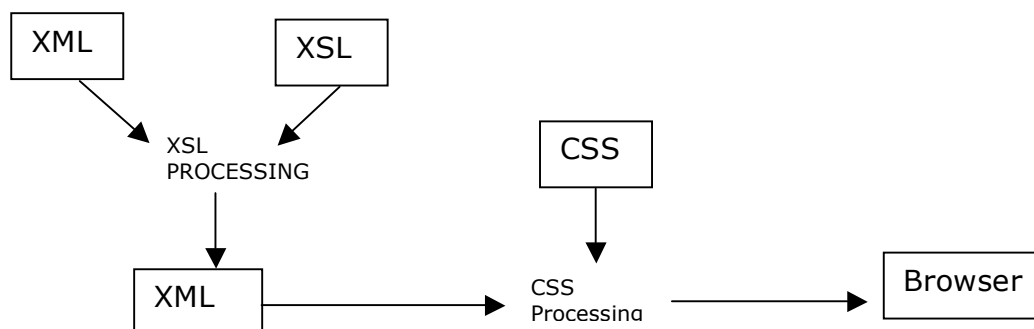
Finally, **XSL Formatting Objects (FO)** ... doesn't matter !

### **XSL vs. CSS**

Both of these select ELEMENTs and apply presentation styles to those ELEMENTs, but ...

<b>XSL</b>	<b>CSS</b>
<ul style="list-style-type: none"> <li>• Powerful but complex</li> <li>• Manipulative</li> <li>• Can change the order of ELEMENTs</li> <li>• Can process ELEMENTs more than once</li> <li>• Can suppress ELEMENTs in one place and display them in another (i.e. can do filtering)</li> <li>• Can add generated text</li> </ul>	<ul style="list-style-type: none"> <li>• Simple</li> <li>• Easy for HTML and XML</li> <li>• Does not support transformations of XML documents</li> </ul>

Diagrammatically ...

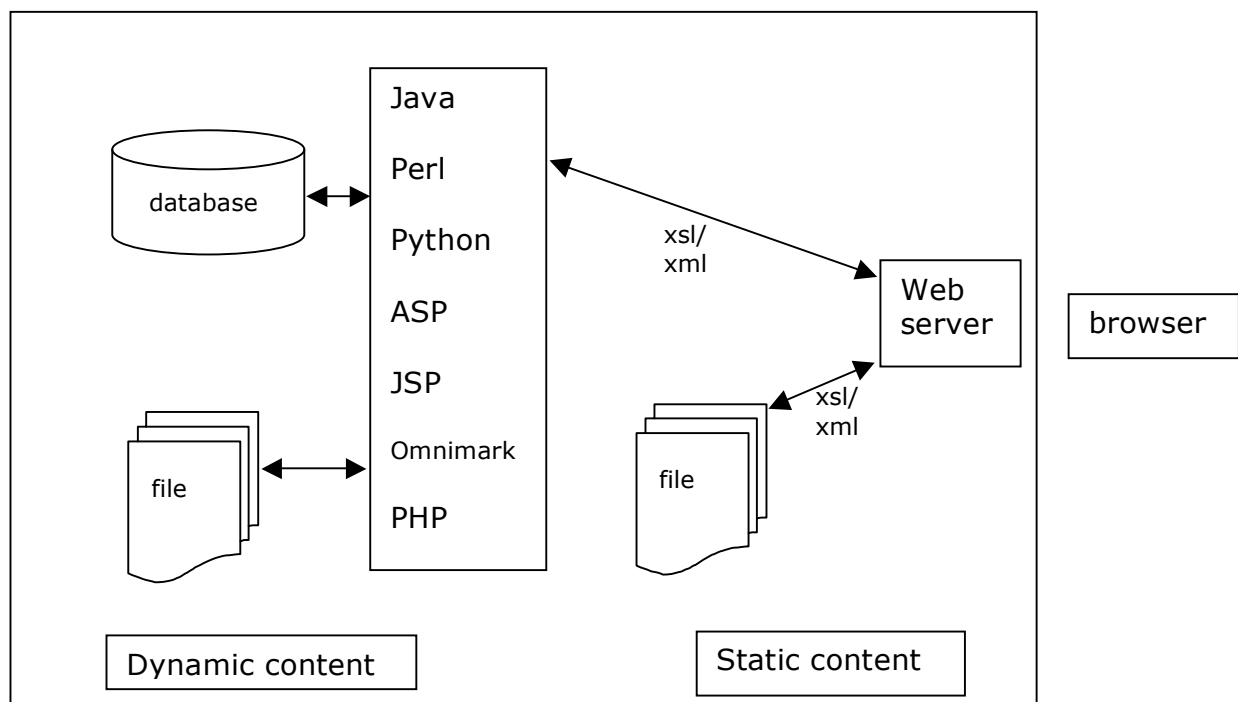


## How can XML be used ?

XML technologies are based around an XML document store, which could be a file system (XML files in a directory) or on a database. If it is a database, then many/most databases, especially Relational databases, now have XML wrappers to map XML "documents" to a relational structure which is not always the most efficient.

So, for example, an XML document describing each student in DCU and consisting of name, course, personal details, exam results, average marks in each module across all students, module specifications, etc., all pooled together from underlying relational tables. Alternatively we are starting to see native XML databases such as Software AG's *Tamino*.

XML document creation can be based on web scripting in Java, Perl, Python, ASP, JSP, PHP, whatever.



The diagram above shows one publishing model but it does not fully illustrate the possibilities of where the processing can be done :

- format processing can be done at the server side to generate device-specific markup from XML and XSL or XML and CSS documents;
- client side processing can be done by the browser (eg IE6) by downloading XML and XSL to the browser, but this does not work for lightweight devices without much computational resources

- client side processing can be done by downloading XML and having XSL from a different source – perhaps the XSL comes from a personalisation engine. A CSS is fairly limited in this respect but an XSL sheet opens up great potential for personalising the delivery of content

The role of Microsoft is worth mentioning here:

- MS announced support for XML in Office 2000 in 1998 with XML based document properties;
- Office XP had SpreadsheetML - using XML with Excel file format ... Office 2003 had even more adoption and it continues today.
- Office Open XML (OOXML) is a specification for the storage of electronic documents, developed by Microsoft Office 2007 product suite and standardized in December 2006;
- A complete break with previous binary based Office file formats, completely new, so not backwards compatible;
- ... a data format capable of being understood across applications, and its open, is a dangerous thing to risk, losing proprietary file formats, and a longer-term investment by MS.
- OpenDocument or ODF, is also a format for describing electronic documents, defined independently and open;
- OOXML and MS have been the subject of debate in the computing industry;
- Some say MS has standardised its proprietary format in order to prevent the widespread adoption of ODF, which could threaten the dominance of Microsoft's Office.
- Competitors will likely implement compatibility with OOXML in their applications, Microsoft has not released any plans to similarly support the ODF.
- Turf wars continue ...

So, **XML** can be good for **B2B data interchange** and for **consistent look and feel to web pages via stylesheets** and **delivery of content to multiple devices** but there is more to XML than its ease of presentation.

**XML** also holds great promise for searching for information – in an ecommerce setting, searching for products and services can be made specific – for general searching, using XML tagsets to improve precision.

What kind of searches could I perform ... certainly much more structured searching and away from the “bag of words” of conventional IR

That’s the plan, anyway.