Web sites from TEI

Sebastian Rahtz

http://www.oucs.ox.ac.uk/oucsweb/

November 2001

The aim of this talk

- To describe how TEI &XML work on the TEI, and OUCS, web sites
- To say why we use them
- To explain how we implement it all
- To demonstrate some proposed features of the TEI web site

Yes, these slides are authored in XML

What XML buys us

By storing our documentation in valid XML, we get:

- Markup free of presentation information
- A proven documentation workflow (SGML history)
- A software-neutral format
- A lot of open-source and commercial tools for processing the material
- A slew of W3C Recommendations on various aspects of the process, including stylesheet presentation

ie breathing space. . .

What use is all that to the TEI web site?

We want documentation which can be

- Put on the web, with a consistent style
- . . . with a different style next week
- . . . with a different style for colour-blind people
- Formatted nicely into a printable PDF file
- Printed out in Braille
- Indexed intelligently by next-generation web indexing robots
- Checked rigourously against syntactic constraints

. . . and all without any human intervention or maintenance cost.

And management too, please

- We want a complete document history and ability to revert to old versions
- We want to know who changed a file, and when, and why
- We want to have a workflow in which documents can be tested, approved, and published

In short, a *Content Management System*.
. . . and remember not to spend any money
. . . and keep it Open Source politically correct

Summarizing the TEI strategy at Oxford

1. Use Perforce change management software to control document history and versioning
2. Web server as Perforce client, synchronizing immediately if checks are passed
3. Two web servers, one for preview, one for publish.
4. Content authored in XML, delivered in HTML and PDF
5. Style is managed by XSLT stylesheets, and CSS attached to the result
6. Stylesheet choice can be set by user

The site is synchronized with Virginia in a separate operation.

So which flavour of XML for a web site?

- There is an XML version of HTML but a clean break and richer vocabulary would be good.
- The Docbook DTD is well-understood, and has rich "computerese" markup
- We could use namespaces, and pull in HTML vocabulary (tables, images, forms, etc): good idea, but removes chance of DTD validation.
- But since we *are* the Text Encoding Initiative, which defines a rigourous markup for textual material, we really ought to use that

Ultimately, it is not a life or death decision, as conversion between DTDs is not a vast undertaking.

The Text Encoding Initiative markup

The TEI has four qualities which are desirable in a multi-author web environment:

1. Exceptionally rich paragraph-level vocabulary

The Text Encoding Initiative markup

The TEI has four qualities which are desirable in a multi-author web environment:

1. Exceptionally rich paragraph-level vocabulary
2. Mature and powerful metadata (the `<teiHeader>`)

The Text Encoding Initiative markup

The TEI has four qualities which are desirable in a multi-author web environment:

1. Exceptionally rich paragraph-level vocabulary
2. Mature and powerful metadata (the `<teiHeader>`)
3. A well-defined and powerful system of modularity and extensions

The Text Encoding Initiative markup

The TEI has four qualities which are desirable in a multi-author web environment:

1. Exceptionally rich paragraph-level vocabulary
2. Mature and powerful metadata (the `<teiHeader>`)
3. A well-defined and powerful system of modularity and extensions
4. Strong and stable documentation

The Text Encoding Initiative markup

The TEI has four qualities which are desirable in a multi-author web environment:

1. Exceptionally rich paragraph-level vocabulary
2. Mature and powerful metadata (the `<teiHeader>`)
3. A well-defined and powerful system of modularity and extensions
4. Strong and stable documentation

Some obvious problems and criticisms

It is not hard to argue that the TEI is a poor choice:

1. The paragraph level vocabulary does not cover web paraphernalia
2. The metadata is not a ‘standard’: RDF and Dublin Core are more practical
3. The TEI is not yet ready for Schemas, or namespaces, which makes genuine modularity (eg replacement table or math markup) hard to achieve elegantly
4. The documentation is not useful for authoring new material

but hopefully we know how to overcome some of the problems...

Which bits of the TEI are needed?

Web authors do not want to be offered a choice of inputting verse, or critical text apparatus. So:

- Start with the ‘prose’, ‘figures’ and ‘linking’ tagsets from full TEI (not TEI Lite!)
- omit 75 elements (eg numbered divs)
- Modify `<table>`, `<address>`, `<figure>`, `<xptr>`, and `<xref>`
- Add 14 new elements
- Can work as a compiled DTD using Carthage or, with the new XML TEI, as a standard modification/extension file pair

There are 128 elements in the resulting DTD, but many of these are in the header.

Modifications to standard TEI elements

- Add some HTML-like border, width etc attributes to `<table>`
- Add *width* and *height* attributes to `<figure>`, and add a direct *file* attribute to avoid predeclaring entities
- Add `<email>` to `<address>` content model
- Add *url* attribute to `<xref>` and `<xptr>` to avoid declaring entities

Additions to the TEI

Largely aimed at inline markup for documenting computer programs, borrowing some features of TEI Lite:

<gi> An SGML, XML or HTML element name eg <h1>

<Button> A button which a user can see eg Logout

<Code> Some sort of computer language code eg \textbf{a}^{34}

<Field> A labelled input field eg Subject

<Filespec> A file or directory specification of any kind eg c:\Windows\My Documents

...continued

<Input> Text for a user to type eg quota

<Key> A key to press eg R

<Keyword> A keyword in some technical code the user is being asked to write eg font-family

<Link> The text of a link which is being described eg IT Information

<Menu> A menu item eg Save as

<Output> What comes back when you give a command eg job completed

<Prompt> A prompt from the computer eg password:

<Value> A possible value for some option eg Times-Roman 10pt

As opposed to a full-scale screen dump, using <Screen>:

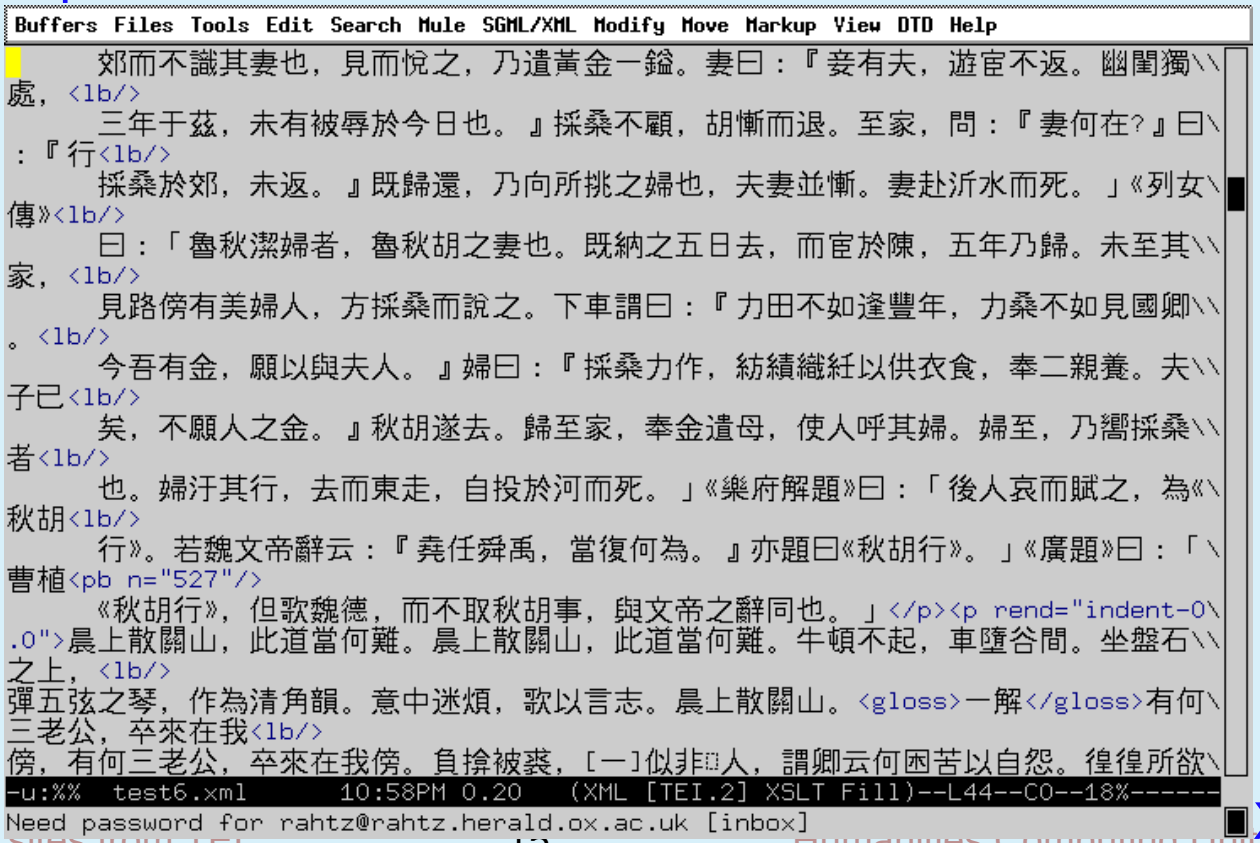
```
rahtz@spqr-tosh:~/oucs$ ls
#sample.xml#   tei-oucs-xml.css  teixlite.dtd
pdfscreen.cfg  tei-oucs.dtd      teixmlslides.sty
pdfscreen.sty  tei-oucs.html     users.ox.ac.uk
psgml.png      tei-oucs.xml      x
sample.xml     teixlite-xml.css  xml-tools.xml
rahtz@spqr-tosh:~/oucs$
```

or <Program>:

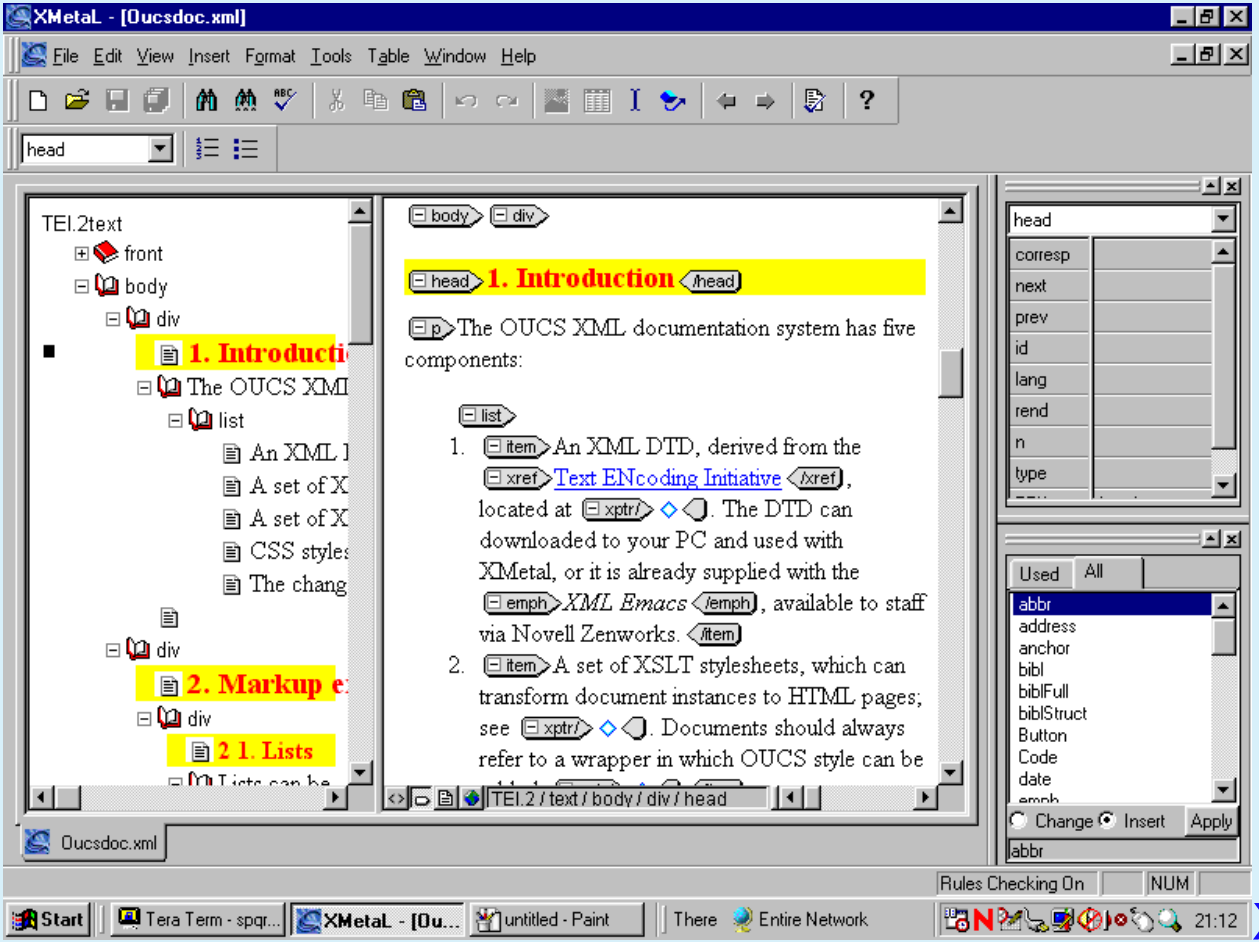
```
#!/bin/sh
# xmltopdfslides. convert an XML file to PDF slides
F=‘basename $1 .xml‘
echo ’\NAME{TEI.2} {teiprintslides.xmt}’ > $F.cfg
echo ’\SYSTEM   {http://www.oucs.ox.ac.uk/dtds/tei-
oucs.dtd}
   {teiprintslides.xmt}’ » $F.cfg
pdfxmltex $F.xml
acroread -toPostScript $F.pdf
```

Authoring environments

This is probably the biggest headache. Who likes authoring in XML? Our only successful experiments are with XMetal, and Gnuemacs:

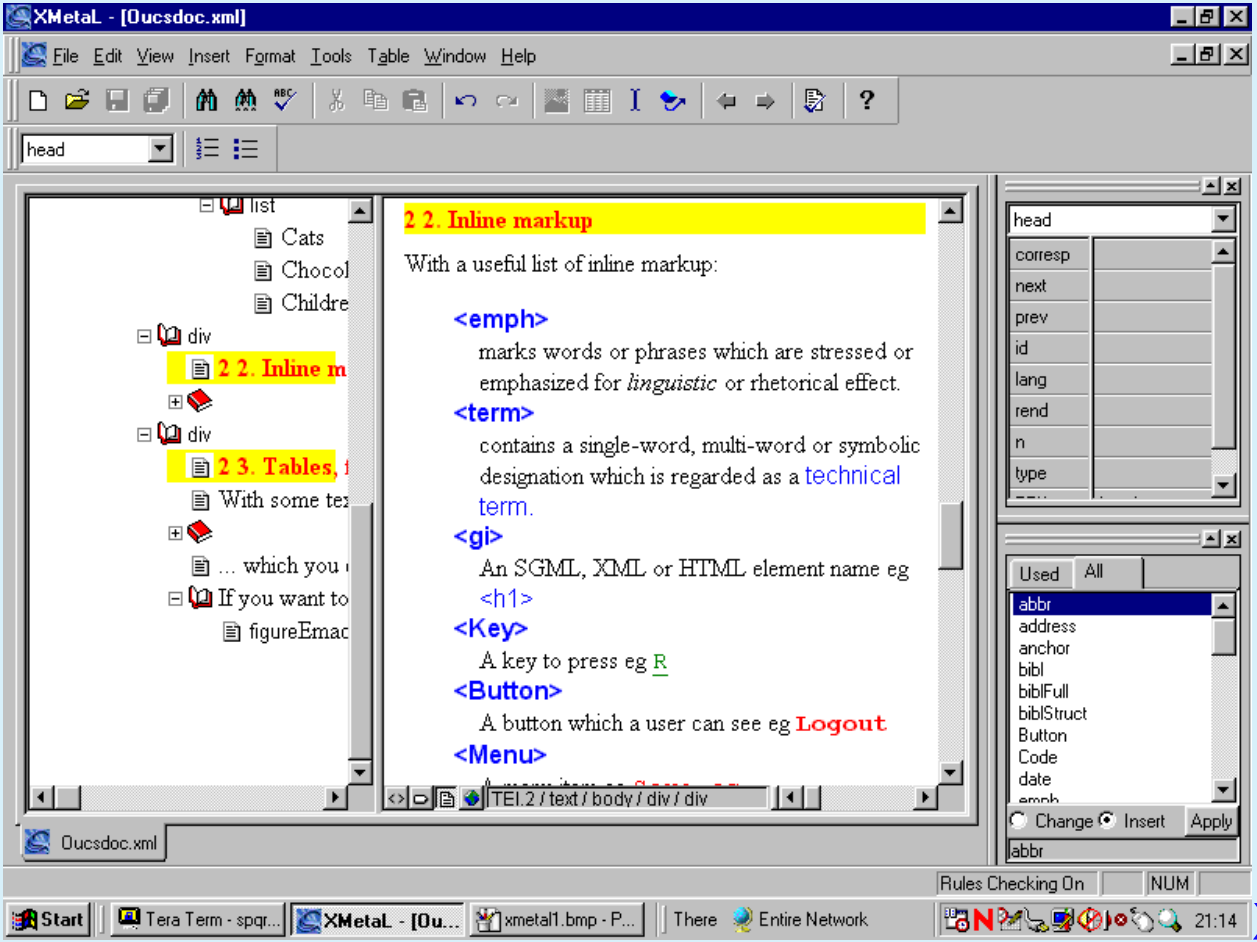


XMetal 1



editing with XMetal, showing view with tags visible

XMetal 2



editing with XMetal, showing plain view with no tags showing

Conversion of HTML to TEI

Inevitably, project-dependent, and messy. Our approach:

- Clean the HTML, and convert to XHTML, using Dave Raggett's *Tidy* program
- Run an *ad hoc* cleanup script (sed)
- Transform to TEI using XSLT specification whose main problem is finding the end of divisions

The same sort of process can be used to convert eg Word to TEI.

Converting TEI back to HTML

We have developed an extensive set of XSLT stylesheets which transform TEI documents to HTML. Although they do not cover the whole of the TEI, they provide a plausible rendition of many documents.

The stylesheets are heavily parameterised using both XSLT variables and named templates.

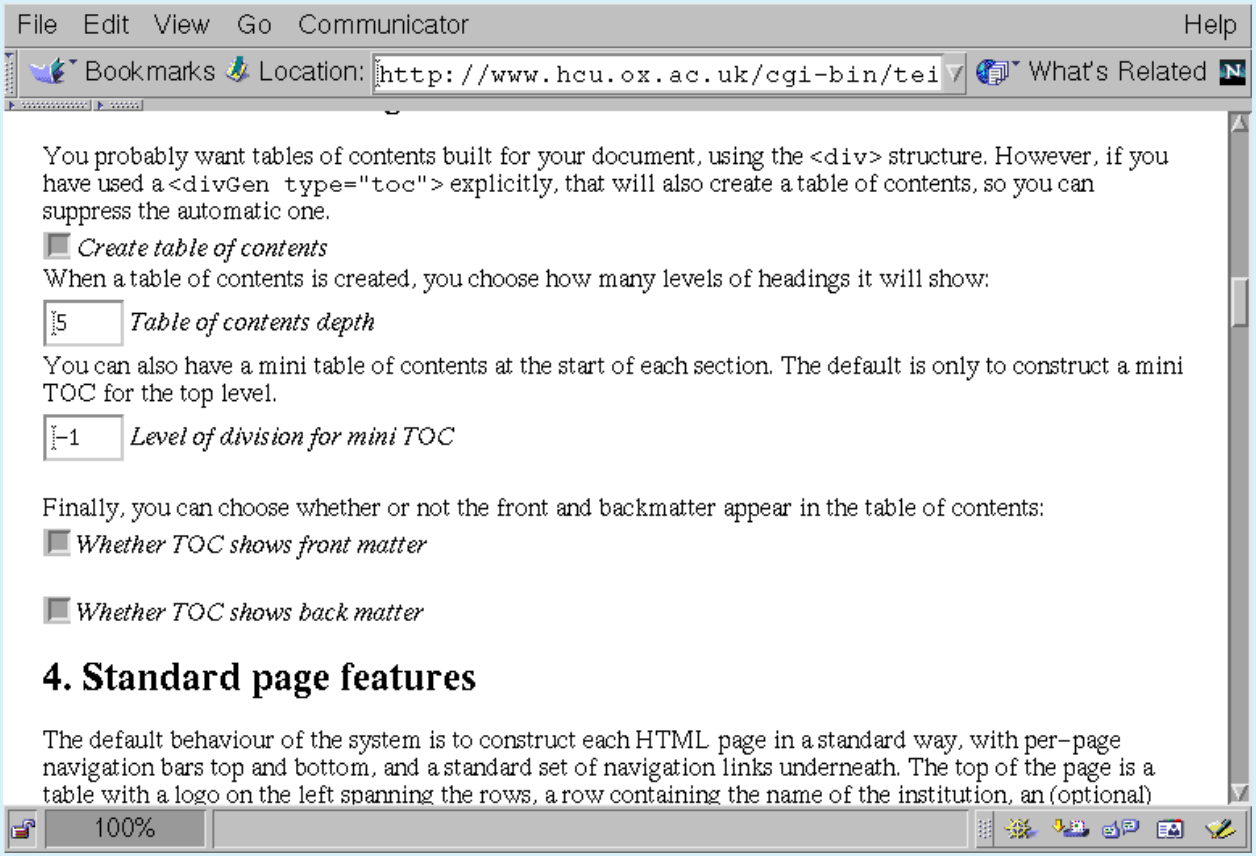
Internationalisation is ongoing, but incomplete.

There are 60 variables and 10 documented named templates (as well as many other undocumented named templates internally which can be overridden).

Key features of this system

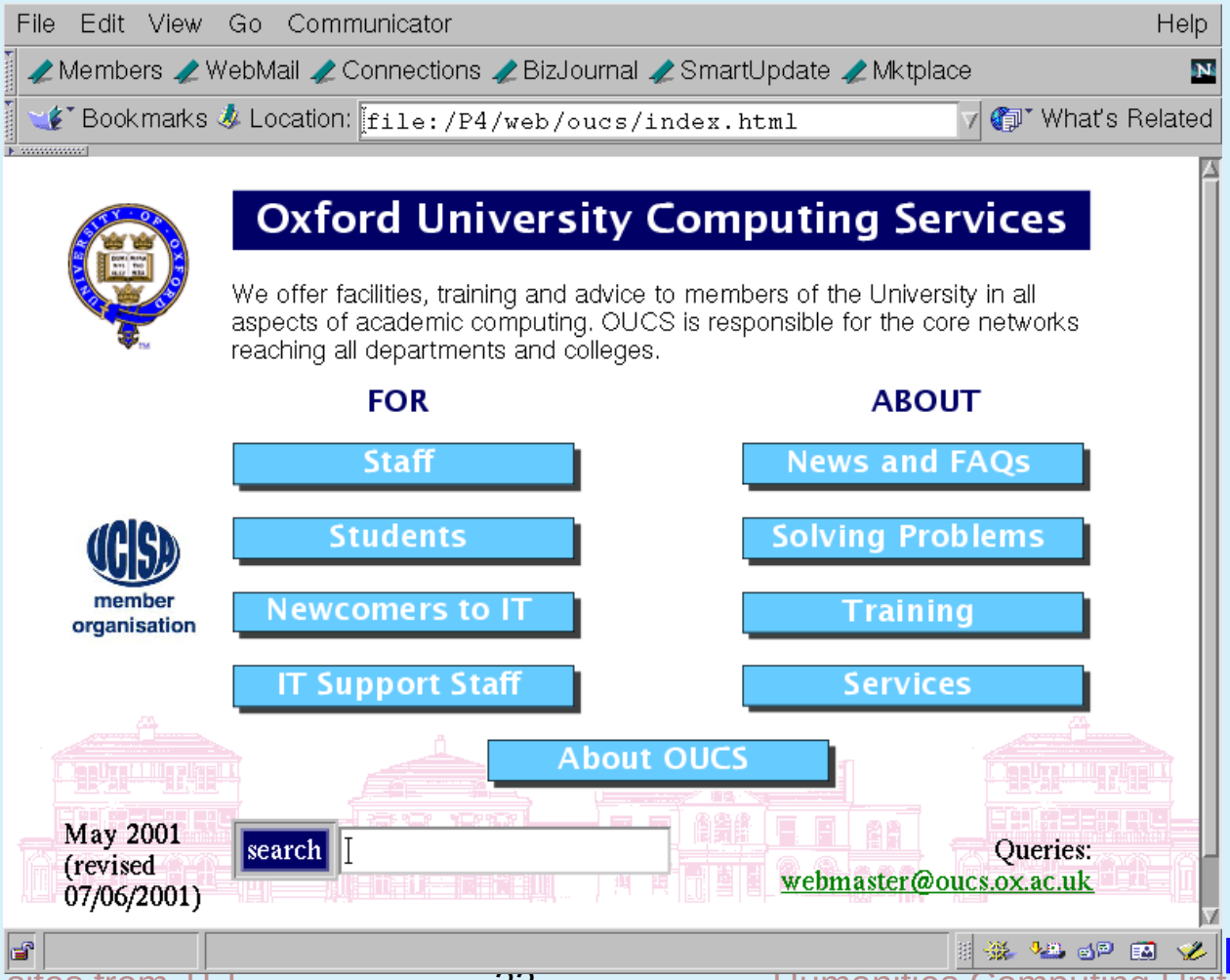
- XSLT stylesheets can import others, and then override selected bits of them
- The local stylesheet can override at any level, right down to trapping character data or `<TEI.2>`
- Local stylesheet wrappers can be created using a web form
- CSS for decorating the HTML gives the designer a second bite at the cherry

Stylebear form



form for creating TEI XSLT stylesheet parameterization

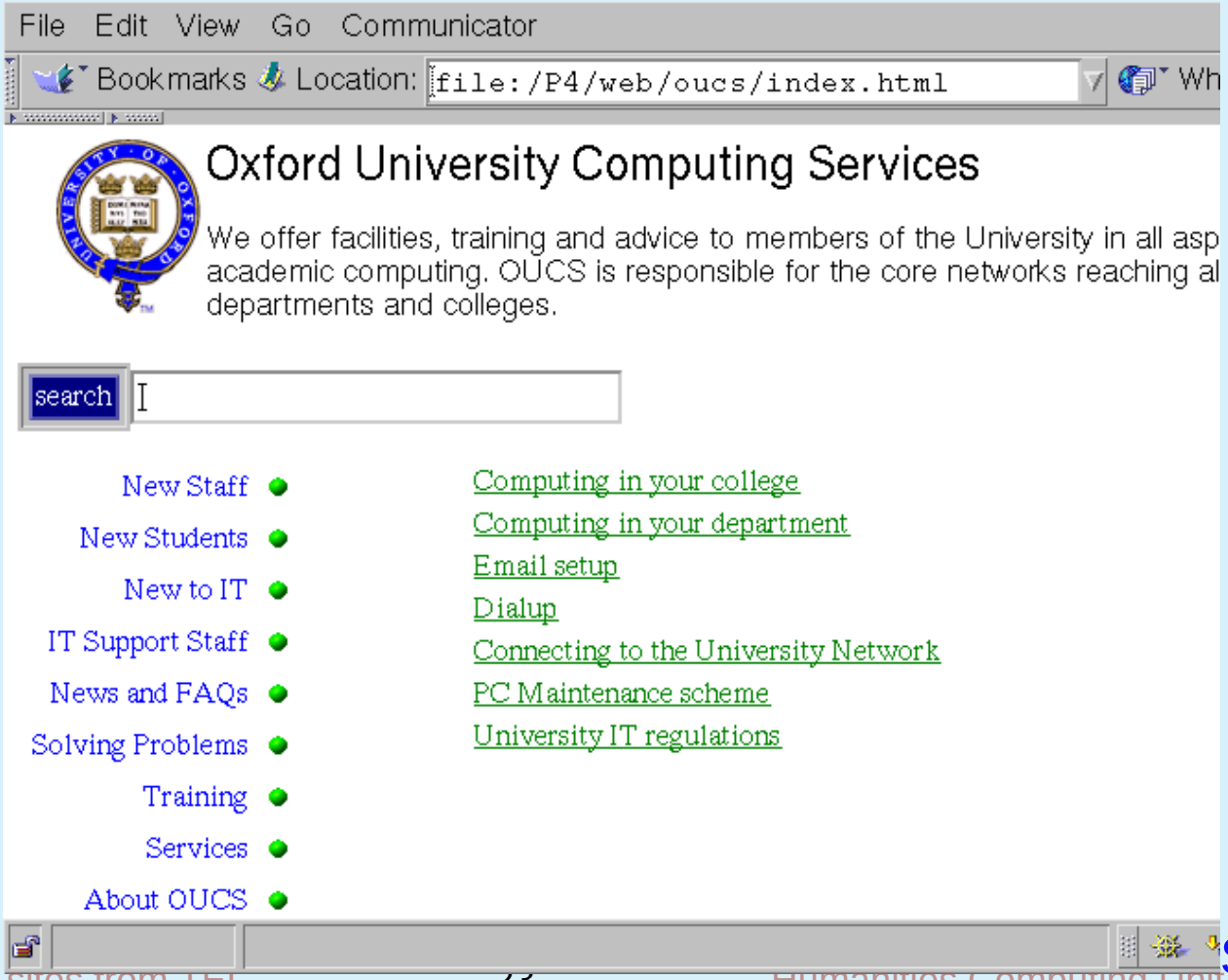
Example rendition I



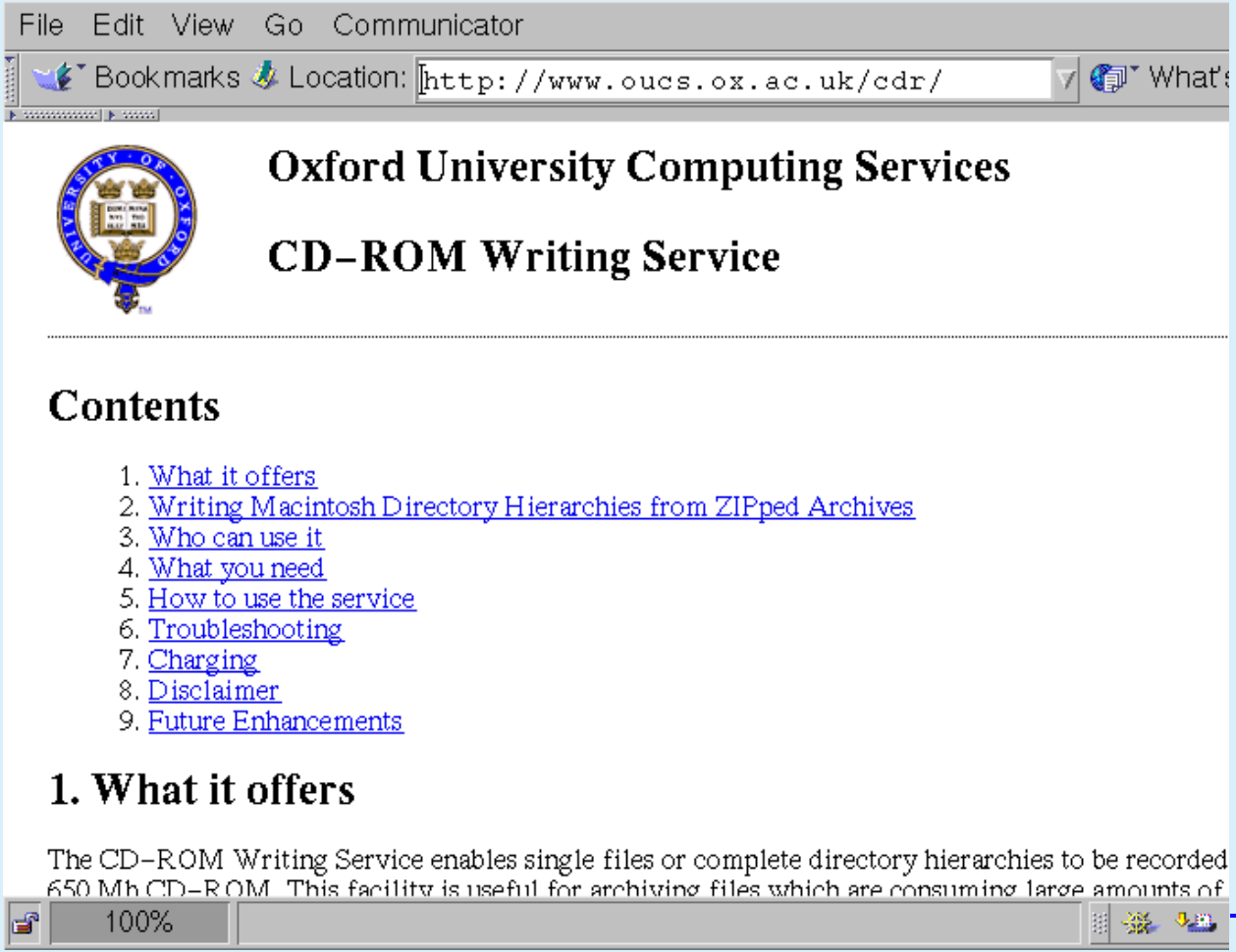
Fi

sample rendition of TEI XML document in OUCS

Example rendition II



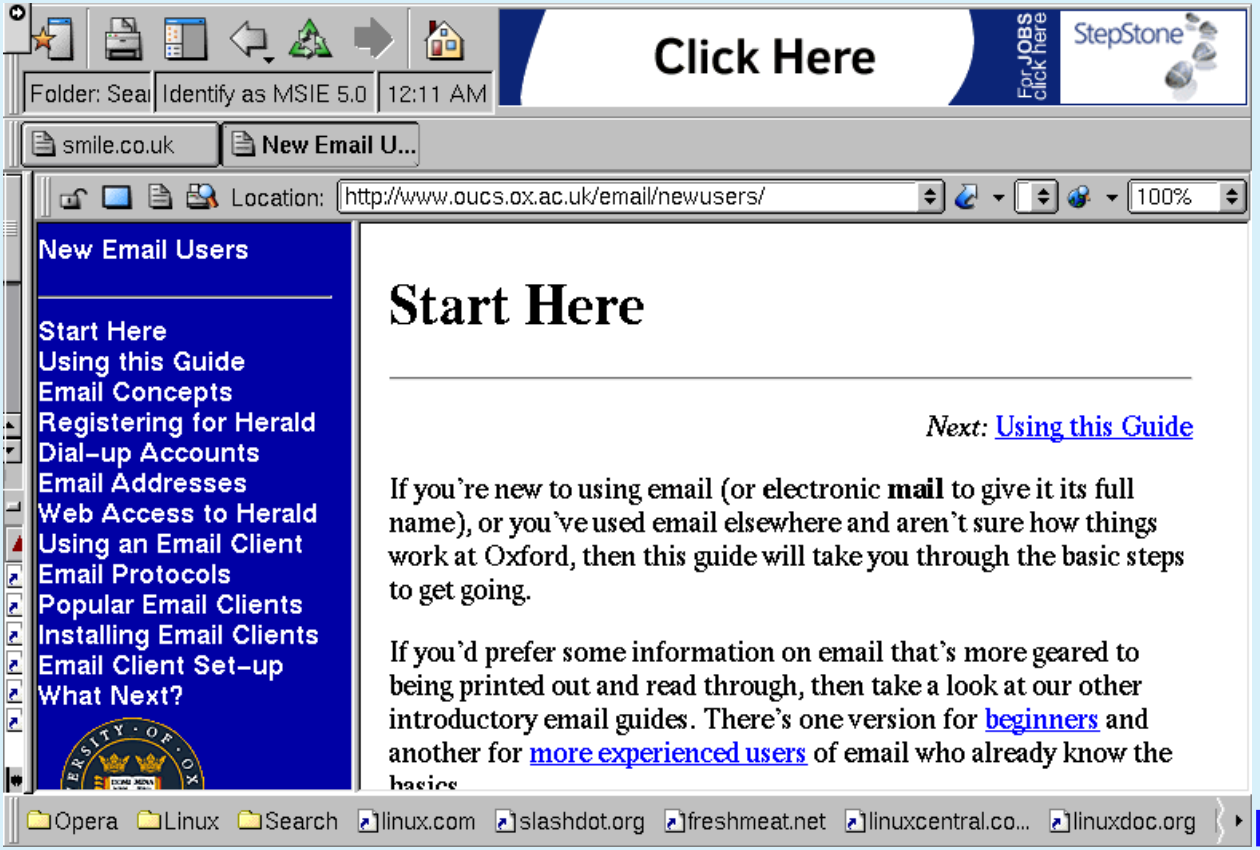
Example rendition III



Th

sample rendition of TEI XML document in OUCS

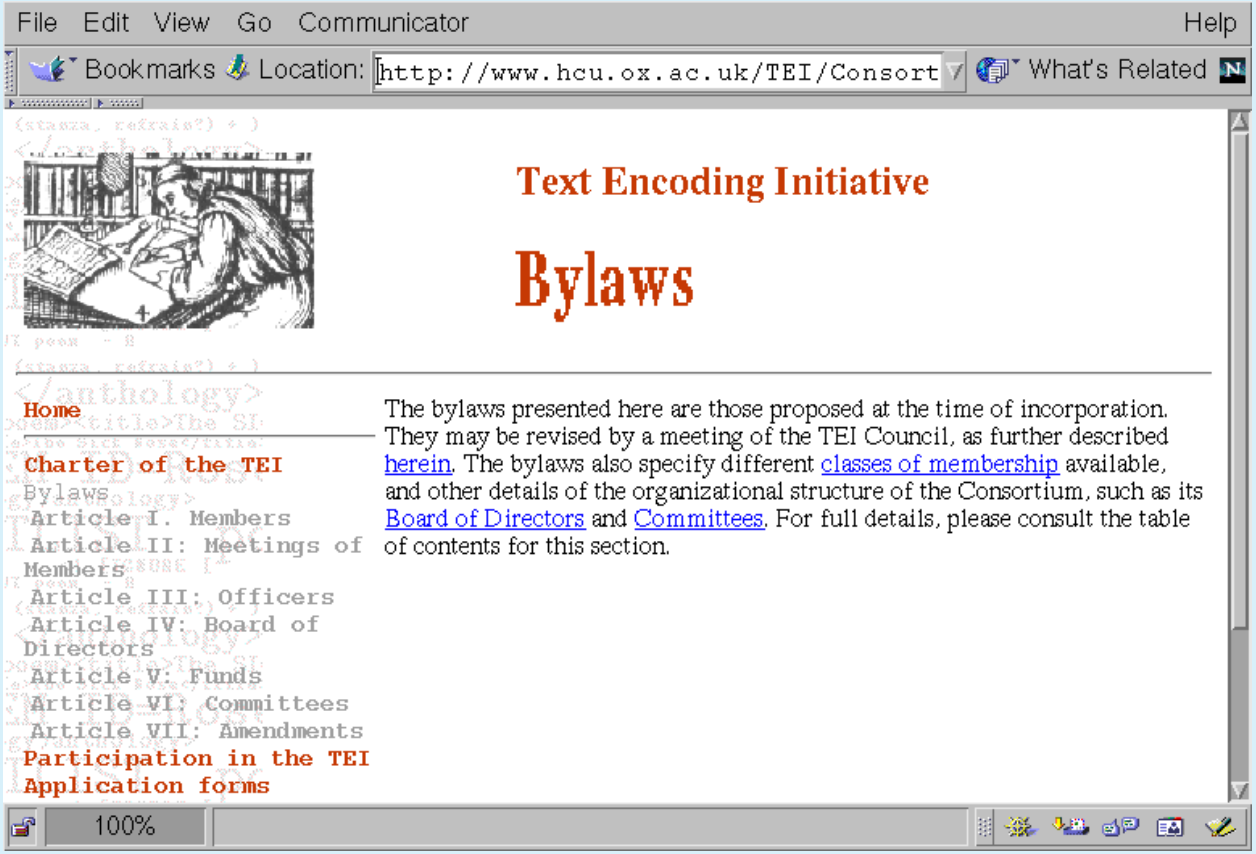
Example rendition IV



Fo

sample rendition of TEI XML document in OUCS

Example rendition V



Fi

sample rendition of TEI XML document in OUCS

XML on the web

Eventually, all web user agents (browsers) will be XML aware! Until they are, we have to choose:

1. transform XML to HTML on the server (statically)
2. transform XML to HTML on the server (dynamically, using a servlet)
3. render XML on the client using CSS, XSLT, or dynamically with some kind of plugin

Production systems: HTML

The OUCS (but not yet the TEI) web site serves HTML dynamically:

- XML files are stored in a change management system (Perforce); the web server keeps in step automatically
- A request triggers an XML to HTML conversion
- Repeated requests come from a server file-based cache
- Style can be changed in various ways

The web server work is done by AxKit.

AxKit

- Open Source package by Matt Sergeant, running under Apache mod_perl
- Uses libxslt XSLT processor by Daniel Veillard
- Normal scripting/extension work is in Perl
- Same philosophy and setup as Apache Cocoon (Java)
- Works with a pipeline of transformations, which may be in XSLT or other languages
- Powerful features for choosing stylesheeting according to PI in source file, DTD, media, client, URL parameters, etc

`http://www.axkit.org`

AxKit Apache configuration example (1)

```
PerlModule AxKit
AxCacheDir /tmp/axkit_cache
AxAddStyleMap text/xsl +Apache::AxKit::Language::LibXSLT

AxAddStyleMap text/xslfo
    +Apache::AxKit::Language::PassiveTeX
AxGzipOutput On
AxDebugLevel 10
PerlHandler \
     Apache::AxKit::StyleChooser::UserAgent \
     Apache::AxKit::StyleChooser::Cookie \
     Apache::AxKit::StyleChooser::QueryString \
            AxKit
PerlSetVar AxUAStyleMap
   "text=> Lynx, \
    css=> Opera, \
    raw=> MSIE, \
    screen=> Mozilla "
```

AxKit Apache configuration example (2)

```
<AxStyleName #default>
  AxAddRootProcessor text/xsl teihtml-oucs.xsl TEI.2
 </AxStyleName>
 <AxStyleName printable>
  AxAddRootProcessor text/xsl
    teihtml-oucs-printable.xsl TEI.2  </AxStyleName>
 <AxStyleName text>
  AxAddRootProcessor text/xsl
    teihtml-oucs-text.xsl TEI.2       </AxStyleName>
 <AxStyleName css>
  AxAddRootProcessor text/xsl
    teihtml-oucs-css.xsl TEI.2        </AxStyleName>
 <AxStyleName raw>
  AxAddRootProcessor text/xsl
    teihtml-oucs-raw.xsl TEI.2        </AxStyleName>
```

AxKit Apache configuration example (3)

```
<Files "*.xsp*">
AxAddStyleMap application/x-xsp
 +Apache::AxKit::Language::XSP
AxAddStyleMap text/xslfo
 +Apache::AxKit::Language::PassiveTeX
AxAddXSPTaglib +AxKit::XSP::Util
AxAddXSPTaglib +AxKit::XSP::ESQL
AxAddXSPTaglib +AxKit::XSP::PerForm
AxAddXSPTaglib +AxKit::XSP::Param
<AxMediaType screen>
<AxStyleName #default>
AxAddProcessor application/x-xsp .
AxAddProcessor text/xsl /stylesheets/teihtml-oucs.xsl
</AxStyleName>
```

Dynamic content

Simplest method is to embed XSP in source file , which is transformed to target XML for ‘real’ stylesheet:

```
<xsp:page
    language="Perl"
    xmlns:xsp="http://apache.org/xsp/core/v1"
    xmlns:esql="http://apache.org/xsp/SQL/v2"
>
...
<p>Run at <xsp:expr>scalar localtime</xsp:expr>
for <xsp:expr>$ENV{’REDIRECT_URL’}</xsp:expr>
</p>
...</xsp:page>
```

Similar to ASP, PHP etc, but note that output goes on to next stylesheet in the pipeline.

More complex XSP: connecting to database

```
<esql:connection>
<esql:driver transactions='no'>Pg</esql:driver>
<esql:dburl>dbname=cem;host=localhost;port=5432</esql:dburl>

<esql:username>www-data</esql:username>
<esql:password></esql:password>
<esql:execute-query>
<esql:query>select numb,forename,surname
from persons where numb &lt; 50</esql:query>
<esql:results>
<table>
<esql:row-results>
<row><cell><id><esql:get-int col-
umn="numb"/></id></cell>
<cell><esql:get-xml column="surname"/></cell></row>
</esql:row-results>
</table>
</esql:results>
</esql:execute-query>
</esql:connection>
```

Production systems: print

We can use the same XML documents to make printed leaflets, user guides (and even slides, like these). This system is based on

- A set of XSLT stylesheets which transform TEI to XSL Formatting Objects markup
- An implementation of XSL FO based on TeX, and the xmltex XML parser (written in TeX)
- The pdfTeX TeX implementation, which generates TeX directly

Remaining problems

- We used to have MathML included, but decided we did not need it. Maybe it should go back
- The inline markup for software documentation is not right yet
- How do we train authors in this vocabulary?
- The TEI community needs a new tagset for authoring
- How do you mix XML and HTML delivery, unless the whole website is one document?