Preservation of word processing documents

Ian Barnes
The Australian National University

Friday, 14 July 2006, 12:50:10 PM

Table of Contents

1. Introduction

Word processing documents are a major problem for digital repositories. As I will explain below, they are not suitable for long-term storage, so they need to be converted into an archival format for preservation. In this report I will address the following questions:

- What file formats are suitable for long-term storage of word processed text documents?
- How can we convert documents into a suitable archival format?

I also address the related non-technical question:

- How can we get authors to convert and deposit their work?

While the vast majority of material generated by universities is text, most research on digital preservation concentrates on images, sound recordings, video and multimedia. You could be forgiven for thinking that this is because text is simple, but unfortunately that's not so. Even relatively short text documents (like this one) have complex structure consisting of sections (parts, chapters, subsections etc) and also of indented structures like lists and blockquotes. A significant part of the meaning is lost if that structure is ignored (for example by saving as plain text).

Most text documents created today are created in a word processor. (The other major text-processing method, used by mathematicians, computer scientists and physicists, is TeX/LaTeX. I will address sustainability of Tex/LaTeX documents in a separate report[1].) For the reasons set out in Section 3.3 below, the file formats generated by word processors are generally not sustainable, so we need to consider converting documents to better formats. Most of the text we're interested in archiving is in one of the various Microsoft Word formats. A small amount is in other word processing formats, notably Open Document Format, which is created by OpenOffice.org Writer and a few other minor word processors.

Since word processing formats are not suitable for preservation, the next question is: "What format should we convert documents to?" Most archives seem to have chosen PDF, but this has serious problems as set out in Section 3.4. XML is a better answer, but it's not a complete answer. XML is not a file format, but a meta-format, a framework for creating file formats. We have to choose a suitable XML file format for storing documents. I discuss this question in Section 3.5.

There are various methods available for converting word processing documents into a suitable XML format. I discuss these briefly in Section 4.

Other people have been thinking about this problem too. In Section 2 I give a very brief literature review, and in Section 5 I list a few case studies of previous practical conversion work, biased towards work done by people I know, here in Australia.

In Section 6 I give a description of my own current work in progress, the Digital Scholar's Workbench, a web application designed to solve some of the problems with preservation and interoperability of word processing documents.

In Section 7 I sum up and make some recommendations.

2. Previous work

There is a lot of published research on digital preservation, but not much of it that I found deals in any detail with preservation of *text*.

There is good work done by the DiVA people in Uppsala University Library, who are archiving documents in XML[2]. They use a custom format which is basically DocBook XML for describing the document itself (content and structure), with a wrapper around the outside allowing for collections of related documents and for comprehensive metadata.

Slats[3] discusses requirements for preservation of text documents, and the relative merits of XML and PDF. Like several other authors with similar publications, she recommends storing documents in XML, but fails to specify *what* XML format to choose.

Anderson et al[4] from Stanford recommend ensuring that documents are created in a sustainable format rather than attempting conversion and preservation later, as I will recommend below. This leaves open the question of what to do with existing documents.

The National Library of Australia[5] recommends converting word processing documents to Rich Text Format (RTF) for preservation. (I disagree. See Section 3.5 below.)

3. File formats

This section is about choice of file formats. First we need to make an important distinction between *preservation* formats and *access* or viewing formats.

3.1. Preservation vs. access formats

A *preservation format* is one suitable for storing a document in an electronic archive for a long period. An *access format* is one suitable for viewing a document or doing something with it.

Note that it may well be the case that no-one ever views the document in its preservation format. Instead, the archive provides on-the-fly conversion into one or more access formats when someone asks for it. For example, the strategy I recommend is to store DocBook XML or TEI, but serve the document up as either HTML for online viewing or PDF for printing.

Some file formats may be suitable for both purposes. XHTML has been suggested, with CSS for display formatting. As XHTML is XML (and particularly if the markup is made rich with use of the `div` element to indicate structure), it may be an adequate preservation format, at least for simple documents. As it can be viewed directly in a web browser, it is eminently suitable as an access format. It does have some shortcomings however, as set out in Section 3.6.3 below.

3.2. Criteria for sustainability

What features does a good preservation format have? How do we judge?

Michael Lesk[6] gives a list of required features for preservation formats. (The points in italics are his, the comments that follow are mine.)

1. *Content-level, not presentation-level descriptions.* In other words, structural markup, not formatting.

2. *Ample comment space.* Formats that allow rich metadata probably satisfy this.

3. *Open availability.* In other words, no proprietary formats. To get a scare, remember what happened to GIF images when Unisys claimed that they were owed royalties because they own the file format[7]. What would happen if Adobe decided to do the same with PDF or Microsoft with Word?

4. *Interpretability.* In other words, the formats should not be binary. It should be possible for a human to read the data, and also for small errors in storage or transmission to remain localised. A small error in a compressed binary file can render the entire file useless.

Stanescu[8] looks at this topic from a risk management point of view. Slats[3] discusses criteria for choosing file formats, coming to very similar conclusions.

3.3. Word processing formats

3.3.1. Microsoft Word

The vast majority of all text documents created today are created in Microsoft Word using its native `.doc` format (in one of its many variations depending on the version of Word being used). It would be great if we could just deposit Microsoft Word documents into repositories and be done with it, but unfortunately that won't do, for a few good reasons:

- Word format is proprietary. It is owned by Microsoft corporation. Even the recent Microsoft Word XML-based formats suffer from this. So why are proprietary formats a bad thing?
    - The owner could choose to change the format at any time, possibly forcing repositories to convert all their documents.
    - The owner could change the licensing at any time, perhaps insisting that documents may only be opened using their software, or that users pay a fee for reading or editing existing documents.
- Except for the recent XML-based versions, Word is a *binary* format. There is no obvious way to extract the content from a Word document. If the document is corrupted even a little, the content can be lost. Even the most recent version, Microsoft Open XML format, is a compressed Zip archive of XML files. Compressed files are particularly prone to major loss if corrupted.
- Word is not just one format but many. One could argue that Microsoft's success has been partly built on making incompatible changes to their format so as to encourage users to pay for new versions of the software. Leaving documents in Word format forces repositories to support not one but several file formats, or alternatively to engage, every few years, in a process of opening *every* stored document in the latest version of the software, and saving it using the most recent incarnation of the format. When the number of documents becomes large, this becomes an unacceptable cost.
- Even the new XML-based format has some technical problems. For example, some of the data in a bibliography entry is stored as strings that need parsing[9], rather than using XML elements or attributes to separate the different items. This makes automated processing of these files much more difficult.

Microsoft has released their latest XML-based file format, known as Open XML[10], publicly, along with assurances that it is and will always be free[11]. Despite the mistrust of many in the open source community,

who remember the GIF/Unisys controversy[12], this appears to be genuine. Nevertheless, there do not appear to be any significant advantages of Open XML over Open Document Format.

3.3.2. Open Document Format

Open Document Format[13] is the native file format of the latest versions of OpenOffice.org Writer[14], the word processor component of the OpenOffice.org open source office suite. OpenOffice.org is the open source version of Star Office, which was originally developed by Star Division in Germany. Star Division were bought by Sun Microsystems, who still support the continuing development. OpenOffice.org is the world's largest open source software project. Most development seems to be done by Sun engineers, but there is also a very active community.

Open Document Format grew out of OpenOffice.org's earlier Open Office XML format. It is now an OASIS and ISO standard and a European Commission recommendation. It is supported by the open source word processors KOffice and AbiWord, with more to come.

An ODF file is a Zip archive containing several XML files, plus images and other objects. The Zip archiving and compression tool is freely available on all major platforms, so there should never be a problem getting at the content of an ODF document. Using a Zip archive does mean that the files are prone to catastrophic loss of content with even minor data corruption, in the same way as the Microsoft Word formats discussed above.

If we are going to archive word processing documents, I believe that ODF is a better option than Microsoft Word format in any of its variations. Even the new XML-based Word formats will still suffer from being owned by a for-profit corporation.

One possible preservation strategy would be to convert all word processing documents to ODF for storage. This can be done easily using OpenOffice.org itself as a converter. The conversion could be set up as part of the repository ingest process so that it would be almost totally painless for users. Conversion to ODF gets all the formatting of most Word documents, with only minor differences in layout. For complex documents that use lots of floating text boxes, these minor differences can make a mess of the appearance of the document. For documents that use embedded active content (chunks from live spreadsheets etc), the embedding will probably fail. For most "normal" documents, even complex ones, the conversion is good.

The main disadvantage of this strategy is that Open Document Format is still a word processing format, not a structured document format. What does this mean, and why is it a problem?

- Word processing formats are at heart about describing the appearance of the document, not its structure. For serious processing it's the structure we want. In 20, 50 or 100 years, most readers will probably not care about the size of the paper, the margins, the fonts used and so on. Even today, if we're going to serve up a document as a web page, those details are irrelevant. Sometimes these details can even be a disadvantage, for example if the document insists on fonts that are unavailable on your computer. On the other hand, the division of the document into sections will always be relevant, useful and important, and must be preserved.

- Word processing formats are flat. That is, the document is a sequence of paragraphs and headings. What we'd really like is a deep structure with sections, subsections and so on, nested inside each other (as in DocBook or TEI). We want this deep structure because it makes structured searches and queries possible, and makes conversion with XSLT much easier.

  It is possible to do automated conversion from flat to deep structure[15] (and see Section 6 below), but this is only possible at the moment with documents that conform to a well-designed template. In the future heuristic methods might extend this to less carefully prepared documents, but the results are likely to be inconsistent.

The other disadvantage of Open Document Format is that even for simple documents it is extremely complex. For example, unzipping a one-page document of about 120 words results in a collection of files totalling 300K in size. This makes it relatively difficult to locate the meaningful content and structure and transform it into other formats for viewing or other uses. Instead of leaving documents in this complex format and having a hard job writing converters (XSLT stylesheets) for all possible future uses, it would be better to store documents in a simple, clear, well-structured format that makes converters easier to write.

3.3.3. Other word processing formats

There are several, but none of them has much market share, nor do any of them have any particularly conspicuous advantages. Probably the best strategy with these is to convert them into Word or Open Document Format, then treat them in the same way as the majority of documents. OpenOffice.org will open many file formats, so it can be used as a generic first stage in any process of converting documents into useful formats. Use OpenOffice.org in server mode to open all documents and save them in Open Document Format, then process them into something better.

3.4. PDF

Many repositories seem to have adopted PDF as their main format for text documents, both for storage and for access. PDF has some good points:

- It is easy to create, either using Adobe Acrobat software or using the PDF Export feature available in both Microsoft Word and OpenOffice.org Writer.
- It can be viewed on all platforms using the free Adobe Acrobat Reader software (with some caveats, see below).
- It is extremely effective at preserving the formatting of a document. For some applications (for example in legal contexts) this may be of vital importance.

However, there are some serious problems with using PDF as a storage format[16]:

- The format is owned by Adobe. While it is currently open, the company could decide to keep future versions secret, charge for use, mandate the use of their software etc. Remember the controversy over Unisys and the GIF image file format[12].
- There are some compatibility problems between different versions.
- Documents may rely on system fonts. There is an option in PDF to embed all fonts in the document, but not all software uses this, and some PDF viewing software either cannot locate the correct fonts or doesn't know how to substitute suitable alternatives. Failing to embed all fonts can result in a serious degradation of the on-screen appearance of a document, or in a complete failure to display the content.

  For example, I recently asked my second-year software engineering students to send me their reports as one-page PDF documents. Most were fine, but a small number of students who prepared their reports using Microsoft Word sent me documents that I could view but not print from Adobe Acrobat Reader on Linux. However the Evince Document Viewer program that came bundled with the Fedora Core 4 Linux distribution printed them perfectly.
- PDF includes extra features like encryption, compression, digital rights management and embedding of objects from other software packages. These all present difficulties, particularly the last.

PDF is an excellent access format for printing to paper. Any good preservation system should be able to generate PDF renditions of documents for this purpose. PDF is not so good for viewing on screen, as it ties document content to a fixed page size. This means that for large page sizes or small screens (e.g. on handheld devices like PDAs or mobile phones) text will either be too small to read or the user will have to scroll back and forth along the lines, which is highly inconvenient. Looking ahead, who knows what viewing formats we will use. We need to be able to reformat content to fit the viewing device.

PDF is not a good preservation format. It is true that some of the problems listed above can be avoided by taking care when creating the PDF. For example, a workflow that ensures all fonts are embedded in documents, prevents proprietary embedded objects, forbids the use of encryption and DRM and ensures that compression is turned off would go a long way toward solving the problems above. It won't solve the issue of Adobe owning the format. It will also be difficult to enforce such a policy when people send in PDFs for preservation and don't want to (or don't know how to) recreate them according to the rules.

3.5. RTF

RTF stands for Rich Text Format. It is a Microsoft specification[17], but they have published it, so one could argue that it is an open standard. It is certainly widely interoperable, with most word processors capable of reading and writing RTF. There are problems with using RTF as a preservation format:

- It is still defined by a corporation, with all the risks that entails.
- There seem to be parts of the specification that are not in the publicly available specification document, and which have changed over the years.
- The specification is not complete and precise, leaving many little quirks.

The National Library of Australia has chosen RTF as its main preservation format[5]. I think a well-chosen XML file format has significant advantages over RTF, but it might well be worth retaining RTF as an access format, since it has good interoperability.

3.6. XML

XML[18] is widely accepted as a desirable format for document preservation. See for example the assessment of XML on the US Library of Congress digital formats web site[19] and the related conference paper by Arms & Fleischauer[20]. The reasons are simple:

- XML is a free, open standard.
- XML uses standard character encodings, including full support for Unicode. This makes it capable of describing almost anything in any language.
- XML is based on plain text. This gives it the best possible chance of being readable far into the future. Even if XML and XSLT are no longer available, the raw document content and markup will still be human-readable. (This will be true even if the *meaning* of the markup has been lost, although formats designed with preservation in mind should make the meaning more or less apparent from the carefully chosen element and attribute names).
- XML can easily be transformed into other formats using XSLT[21].

This last point is very important. It means that documents which are stored in XML can be viewed in multiple formats. A minimal solution would generate HTML for on-screen viewing and PDF for printing.

However, just saying “XML is the answer” isn’t enough. Unfortunately this seems to be as far as most of the literature I’ve seen goes. XML on its own is little better than plain text. What makes it useful is when documents conform to a standard DTD or schema. Having an XML-based preservation strategy means choosing one or more (but preferably very few) XML document formats. It also means having a workable method for converting documents into that format.

To summarise, if we decide to convert word processing documents into XML for archiving, this raises two issues:

- What XML format(s) to target, and
- How to do the conversion.

I address the first issue in sections 3.6.1–3.6.4 below, and the second issue in section 4.

3.6.1. DocBook XML

DocBook[22] is a rich and mature format that has been in use for about 15 years. It was originally an SGML format designed for marking up computer documentation (like the O’Reilly books), but its application is wider, although it still seems a bit awkward and ill-matched to non-technical writing. DocBook is an OASIS[23] standard.

DocBook is huge, with over 300 elements. This makes it quite hard to learn, and cumbersome to use directly. Very few people create DocBook documents by hand. Of course that’s of no concern to the ordinary author if the transformation from word processor formats to DocBook is done automatically. It *may* be a concern for the unlucky person who has to write stylesheets for converting documents to and from DocBook. Fortunately though, Norm Walsh (the guiding force behind DocBook) and others have written a comprehensive set of XSLT stylesheets[24] for converting from DocBook XML into numerous formats including XSL-FO (and hence PDF), XHTML, HTML, HTML Help, Java Help, Eclipse Help and so on. This is a huge headstart.

For converting word processor files to DocBook, the complexity and number of elements doesn’t matter, since the conversion process will probably target only a small subset of DocBook. This is the approach I have adopted with the Digital Scholar’s Workbench (see Section 6 below).

3.6.2. TEI

TEI stands for the Text Encoding Initiative[25]. Its guidelines are aimed mostly at the preservation of literary and linguistic texts (so a very different slant to DocBook). Like DocBook, TEI is huge. Furthermore, it’s not exactly *a format*, but a set of guidelines for building more specialised formats. One such is TEI-Lite, which has proved very popular, and is used by several serious repositories.

TEI may be better-matched than DocBook to some scholarly work, particularly in the humanities. It does have some serious shortcomings however:

- It uses abbreviated element names like `<p>` for paragraph (where Docbook uses `<para>`). This is presumably to make it easier to key in by hand, but it is a problem for sustainability since it may make it more difficult to recover the meaning of the markup in the distant future.
- It has a set of customisable XSLT stylesheets written by Sebastian Rahtz[26]. I have no experience with

using them but the impression I get is that they are less mature and less comprehensive than the Norm Walsh DocBook XSL stylesheets[24]. This is definitely worth further investigation.

Whether or not the TEI XSLT stylesheets are up to the job, TEI needs to be considered as a serious candidate for a preservation format for some scholarly writing. Ideally a full solution to the preservation problem would support both DocBook and TEI, allowing authors or curators/archivists to choose the most suitable format for preserving each work (or collection of works).

3.6.3. XHTML+CSS

Since XHTML[27] is both a valid XML document format and can be displayed by web browsers without transformation, with the formatting controlled by a CSS[28] stylesheet (embedded or external or a combination of both), this has been suggested as a possible archival format.

I don't recommend it, except perhaps for low-value documents that archivists cannot afford the time needed to get into DocBook or TEI. In these cases a reasonable strategy might be to store the document in Open Document Format and add an automatically generated, perhaps poor quality, XHTML+CSS version for easy viewing and searching. This could either be stored in the repository alongside the ODF version, or could be generated on the fly by a front-end like Cocoon[29].

Why not use XHTML+CSS for all documents?

- Firstly, it's essentially a flat format which means it's harder to do useful conversion into other formats in the future. It's possible to use the `<div>` element creatively to add lots of structure, but if you're going to do that, you're much better off using a well-defined structured format like DocBook or TEI. (Why? Because in those formats the structural elements are rigorously defined, while in XHTML you can use divs however you like, making it hard for processing applications to know what to do. See Section 3.6.4 on Custom schemata below.)
- CSS relies on consistent use of the "class" attribute in the XHTML. There is no standard for doing this. Same problem as above.
- CSS is not XML, so parsing it to convert it into some new format in the future is much harder than with XML formats.

3.6.4. Custom schemata

One of the biggest traps in the XML world is the idea that you create your own document schema that perfectly matches your particular needs. A university could create specialised document types for lectures, lab exercises, reading lists, research papers, internal memos, minutes of meetings, rules, policies, agendas, monographs... The list goes on. There are serious problems with this approach, as Tim Bray points out[30]. For sustainability, the problems basically narrow down to maintenance and interoperability.

The first problem is maintenance. Each of these formats will require stylesheets for rendering into whatever viewing formats are needed. A reasonable short list would be HTML, PDF and plain text. That's three stylesheets for each document type. What happens next is that someone wants to add an element to one of the document types: a mathematician wants to use maths in a meeting agenda, for example. Every time this happens, you have to modify all the stylesheets for that document type. With some care in the design, there will be elements that are common to the different document types, and it may be possible to do some sharing of templates, but in general the workload is large and ongoing.

The second problem is that you lose interoperability. One of the long-term goals of the whole repository project is that one should be able to retrieve a chunk of something from the repository, and drop it into another document of a different type. The use of custom schemata acts against these goals. We'd also like people elsewhere to be able to use the documents that we go to so much trouble to preserve in our repository, but we can't expect them to know all about our special document types. So then we would have to create more stylesheets for exporting the documents in well-known interchange formats. The maintenance nightmare grows. Not recommended.

I prefer a strategy that involves one set of styles, designed to accommodate all the document types. Peter Sefton agrees[31]. You *might* want more than one template (but try to keep it to a minimum), perhaps including boilerplate text or with the headings already set up for particular types of documents, but the bottom line is that they must all use the same small, generic set of styles. The styles need to be designed so that you can automate the conversion to a generic structured XML format like DocBook or TEI. That's what gets stored in the repository, and served up (perhaps via Cocoon) in various formats. This drastically reduces the maintenance problem. The more authors can be kept within a restricted basic set of word processing features, the better. One strategy is to only guarantee conversion for documents that follow the template; go outside that list of features and you lose the high-quality conversion and the range of output formats.

4. Converting documents into DocBook (or TEI)

Having decided on a suitable archival format, the second issue is how to convert documents into that format. At a first approximation, this is solved by using OpenOffice.org to convert multiple formats into Open Document Format, then unzipping the result and applying one or more XSLT transformations to the pieces. For some processing, XSLT can be quite cumbersome, and direct manipulation (for example using DOM and one of the various DOM bindings: Java, Python, Perl...) can be an alternative worth considering. (See the case studies in the next section.)

The only drawback of DocBook (and the same applies to TEI) is that most word processing documents do not contain enough structure information to allow for an automated conversion. In order to convert word processor documents into DocBook (or TEI), some human effort is required:

- The best scenario is that the document was created using a well-designed template, so that every paragraph has a style name attached to it. These styles can then be used as hooks by an automated conversion process in order to deduce structure. The USQ ICE project[32] (discussed in Section 4.2 below) is a successful example of this approach. This is also working in the Digital Scholar's Workbench (see Section 6 below).
- For legacy documents or authors who refuse to use a template, the word processing document will have to be edited by an electronic archivist to get it into a state where it can be converted to DocBook (or TEI). The obvious way to do this is to open it in a word processor, import the template, and then go through the document applying the template styles. With a bit of help from keyboard shortcuts and macros (available in both Word and Writer), this might not be too painful. If there are a lot of documents to convert, it might prove more efficient to convert the documents to unstructured XML and then clean them up in an XML editor. (Of course this would require trained staff.) Another possibility is to create a specialised electronic archivist's workbench application for doing this kind of work.
- For documents that are extremely poorly formatted, or that exist only on paper, another alternative is to send them out to be rekeyed. This is expensive, but for high-value documents or for small projects it may be worth it. A few thousand dollars for typing and marking up a book may compare well with the cost of setting up the infrastructure to do automated conversion, training staff to do the technical editing (cleaning up the markup, making it conform to a template) and so on. One important possibility worth investigating here is of having documents re-keyed in Word using a good template, and then converting to DocBook

automatically. A first inquiry about this suggests that it costs roughly three times as much to mark up text in DocBook XML as it does to rekey it in Word. That means we can potentially save two-thirds of the cost if we do the conversion to DocBook with an automated process[33].

5. Case studies

5.1. ACS ePub, xPub and predecessors

OpenOffice.org can run in “server mode” where it waits for connections from other programs that instruct it to carry out various operations. Programs can be written in any language that has a binding to the OpenOffice UNO (Universal Network Objects) API. Supported languages include C++, Java and Python (but there may be more by now).

In 2003, following a suggestion from Tom Worthington, I used some sample Java code from the OpenOffice Software Developer’s Kit to open Word documents that had been written using the Australian Computer Society’s journal article template, and save them in the OpenOffice .sxw format. This was followed by applying one or more XSLT stylesheets to transform the documents into clean, simple XHTML.

In 2004 Tom and I supervised Tim Wilson-Brown, a third-year ANU software engineering student, for a short research and development project. Tim extended my earlier work and created the xPub online document conversion tool[34]. The main technology used in this project was PHP. xPub is available under the GPL.

In 2005, Tim led a group of 3rd and 4th year students in their full-year team software engineering project, with Tom and the ACS as clients, developing the conversion and formatting component of a full electronic publishing system, taking Word documents that conform to a template, and producing both good PDF output for printing and good HTML output for the online version of the journal.

5.2. First

Each year the School of Creative Communication at the University of Canberra produces *First*, an anthology of student writing[35]. I did the technical production for the three years 2003-5. The process is:

1. Use OpenOffice to convert around 30 Microsoft Word documents into the old OpenOffice `.sxw` format (the predecessor of Open Document Format).
2. Use `unzip` to extract the `content.xml` file from each `.sxw` file.
3. Use (a few stages of) XSLT to convert each document into a simple custom XML language and from there to LaTeX.
4. Typeset LaTeX to PDF using the PDFTeX program.

This has been quite successful. We shouldn’t rule out the possibility of using TeX/LaTeX/PDFTeX for rendering XML to PDF. It is a more mature technology than XSL-FO at the moment, particularly as far as free implementations go.

5.3. ANU ePress

Brendon McKinley, formerly of the ANU ePress, has been doing XML-based publishing for several years. The focus is on high-quality PDF output for creating books.

Brendon experimented with using OpenOffice to convert Word documents into XML. He now uses UpCast, a commercial product[36]. UpCast converts from RTF to XML and runs cross-platform. If you are running Windows and Word, UpCast has an extension (basically a Word plugin) that allows you to convert directly from Word to XML. Like OpenOffice, UpCast has an API that allows you to drive it from software. Brendon's experience is that that UpCast API is cleaner and easier to use, and that UpCast conversion gives better fidelity to the original document. Of course the disadvantage is that UpCast is commercial software, so it would probably not be feasible to deploy it to every desktop in the university. Site licences are available but are expensive and still seem restrictive.

Here is a typical workflow at the ePress, for publishing the proceedings of a conference:

1. Receive around 15 unstyled Word documents.
2. Editor does content-editing and adds some styles.
3. Brendon adds more styles to indicate the document structure.
4. Use UpCast to convert to DocBook XML.
5. Use the Norm Walsh stylesheets, heavily customised, to produce XSL-FO (Extensible Stylesheet Language Formatting Objects, an XML language for expressing page layout[37]).
6. Use the XEP FO processor (another commercial product, by RenderX[38]) to render the XSL-FO as PDF. (They tried using the open source FO processor FOP, but found that it sometimes breaks on the complex FO generated by DocBook and Norm Walsh. It also doesn't handle tables well.)
7. Use the Norm Walsh stylesheets, unchanged, to convert the DocBook to HTML, and use CSS to customise the appearance.

In the future, the ePress plans to add depositing the DocBook XML versions of their publications in the ANU's digital repository to this workflow.

For book publishing, the quality of the typeset output from this sort of workflow is marginal. Traditional publishing people aren't completely happy with it. The ePress uses B5 paper rather than A5 so as to give the FO processor more scope in finding good line breaks. They also print on thicker paper than most publishers, to compensate for not setting on a grid. Presumably, over time, FO processors will improve.

Brendon believes that the complexity of DocBook is a significant obstacle to doing this sort of work. For most of their projects, however, the full complexity of DocBook is not needed. He suggests that it might be worthwhile to develop a simplified version of DocBook aimed at simple prose. (A bit like TEI Lite perhaps?)

5.4. USQ ICE project

The University of Southern Queensland has many years of experience delivering distance education, first through printed course "bricks" and more recently also over the web. Peter Sefton[39] is leading the development of their latest effort, called ICE[32], which builds on this experience.

Authors create documents in a word processor (either Word or Writer), using a generic template. They must use styles, and only the special styles in the template, not the standard built-in styles. The key to effective web publishing like this is to have a fast feedback loop. Instead of authors sending their work to a web publisher and getting the result back weeks later, they save their document and click “Refresh” in their browser to see the results. If they have done something wrong, they see it straight away.

Some of the messier processing is done using Python. There’s also a lot of XSLT. The system uses Subversion to give authors version control on all documents it manages. (This lets you roll back changes to any previous version at any time.) It also lets multiple authors collaborate on documents. At the moment the closest thing to an archival format produced is XHTML, but we’re talking about maybe integrating with my work (see below) on converting ICE documents into DocBook.

5.5. National Archives Xena project

Xena[40] stands for XML Electronic Normalising of Archives. It is a preservation tool developed by the Australian National Archives[41]. With most resources it simply recodes the contents in Base64 and wraps the result in XML, including some metadata.

With Word documents, Xena uses OpenOffice.org to convert them to Open Document Format. They are making no attempt to take documents out of this word processing format (flat structure, visual formatting) and into a structured XML document format like DocBook or TEI.

Xena is a Java GUI application, complete with menus and toolbars and so on. This infrastructure is expensive to set up, but relatively easily extended. One possibility might be to use the existing Xena software as the basis for an “electronic document archivist’s workbench”. (See below.)

6. The Digital Scholar’s Workbench

A large part of my time and energy in this project has been devoted to constructing a prototype application — The Digital Scholar’s Workbench[42][43] — that implements some of the ideas in this report. The Workbench is a web application that converts suitably structured word processing documents into archival quality DocBook XML[44], and then from there into XHTML for onscreen viewing and into PDF for printing.

In order to work with the workbench, documents must be written using the USQ ICE template[32]. To many people, this seems like a major restriction, and a very common response is that people simply won’t do that. Experience shows however[45], that authors *will* work with a template if:

- it is sufficiently rich to capture their documents without restricting them too much; and
- they can see the benefits in terms of time saved wrestling with their documents and trying to convert them into other file formats for publication.

This approach is backed up by Liegmann from Die Deutsche Bibliothek, who states that:

> All of us presenting papers at this conference have experienced that you need to tolerate and to adhere to a structured framework in order to profit from its advantages.[46]

Further development of the Digital Scholar’s Workbench will focus on making its support for Word and Writer documents more robust and interoperable, adding support for TeX and LaTeX documents, improving the links

to repository software, adding one-click publishing to blogs, websites and learning management systems, support for complex multi-part documents like books and theses, integration with desktop publishing software to produce high-quality typeset PDF output, improved metadata entry and storage, linking with a version control system, round-tripping of documents to enable seamless collaboration between co-authors using different word processing software, platform- and software-independent bibliography management (perhaps outside the limited and hard-to-use systems built in to Word and Writer) and support for presentation slides. The prototype workbench will soon be made available to developers and early adopters as an open source software project, probably through SourceForge[47].

The Digital Scholar's Workbench is built on open-source technology. It uses the Apache Cocoon[29] web application framework, which incorporates the Xalan XML parser[48], the Xerces XSLT processor[49] and the FOP XSL-FO processor[50]. It also uses OpenOffice.org in server mode to transform Word documents into Open Document Format. It relies on the USQ ICE template.

For more information about the Digital Scholar's Workbench, look for a forthcoming article[51].

This document was written in OpenOffice.org Writer and converted to HTML and PDF by the Digital Scholar's Workbench.

7. Conclusion

7.1. Recommendations

1. *Do not use commercial document formats for archival storage.* This means no Microsoft Word, format, no RTF and no PDF. We must store documents in a free format to avoid potential legal problems of ownership and access.

2. *Avoid reliance on commercial software.* This means no UpCast and no XEP, unfortunately. If one of the fundamental aims of the entire digital repositories effort is democratisation of knowledge, then this is incompatible with reliance on commercial software. However it might be wise to allow for drop-in replacement of free open-source components with commercial equivalents. So for example, if an organisation can afford a licence for XEP, they can configure the system to use it instead of FOP, and perhaps (at least in the short to medium term) get better print output.

3. *Documents should ideally be stored in a structured format rather than a visual format.* This means converting word processor documents to DocBook or TEI, or for lower-value documents, XHTML, rather than storing them in a word processor format.

7.2. Proposed preservation strategy for word processing documents

We need to distinguish between legacy documents and documents that haven't been written yet.

1. For documents that haven't been written yet, the best strategy is to have authors create them in a way that makes preservation easy. This means working with campus IT trainers so that in their basic word processing course they give people a good template and teach them how to use it. An environment like the Digital Scholar's Workbench can then allow authors to convert documents into the archival format by themselves.

2. For new students and staff this may be enough. For people who already “know” how to use their word processor, we need to give them an incentive to change their ways. This is why providing a system that gives a range of benefits will have better success. If authors have to learn a new way of working just so they can stick their work in the archive, it’s unlikely to happen. But if learning a new way of working gives them easy upload to course websites, blogs etc, easy formatting of articles for submission to different journals, easy interfacing with backup and version control, *plus* easy deposit into the digital repository, *then* perhaps people will be prepared to change. This is the reasoning behind the Digital Scholar’s Workbench.

3. For legacy documents, and for new documents written by authors who (for whatever reason) don’t adopt the template and styles, preservation will be harder. For high-value documents, some human being will have to put work into them in order to raise them to a standard suitable for archiving. Perhaps this means opening them in a word processor and applying the template and styles. Perhaps it means using some new custom-built “digital document archivist’s workbench” software—some integrated combination of an XML editor and document converter and a bunch of other tools. This work will require specialised staff. The injunctions against complex hard-to-learn software (like XML editors) and against commercial software (like UpCast) need not apply here, since this will only be used by a small group of specially trained people.

4. For high-value documents that are either only available on paper or extremely poorly formatted, re-keying is an option. Note that there appears to be a very significant difference in price—roughly a factor of three—between re-keying directly as DocBook XML and re-keying as a word processor document. If the re-keying service can be convinced to use the template and to produce well-structured word processor documents suitable for automated conversion to DocBook XML using the Digital Scholar’s Workbench, this could save a great deal of money, making re-keying a viable option for a larger number of documents.

5. For lower-value documents, it won’t be worth putting in all that effort, but we might still want to preserve them. In this case, applying a simpler conversion process that converts a word processing document to an equivalent (unstructured) XHTML document may be a good option. Such conversions exist (and are better than those provided with word processors). There is one included in the ICE distribution[32].

6. At least in the short to medium term, it would probably make sense to archive the original word processor document alongside the XML version, just as a backup until the integrity of the conversion process is assured. Eventually these backup copies will become obsolete and unusable, but it will be good to have them there in the meantime.

8. Acknowledgements

This work was funded by the Australian Commonwealth Department of Education, Science & Training, through the Australian Partnership for Sustainable Repositories, which is part of the Systemic Infrastructure Initiative, part of the Commonwealth Government's “Backing Australia's Ability—An Innovative Action Plan for the Future”.

I thank the Australian National University’s Digital Resource Services program, part of the Division of Information, for hosting me while I did this work: Program Leader Peter Raftos, APSR Project Leader Adrian Burton and my colleagues Chris Blackall, Leo Monus, Scott Yeadon, Margaret Henty and Fiona Nelson Campbell. I also thank the ANU Department of Computer Science for releasing me from teaching while I did this work.

References

[1] Ian Barnes, *Preservation of LaTeX documents* (2006) In preparation.

[2] Eva Müller, Uwe Klosa, Peter Hansson, Stefan Andersson & Erik Siira, *Using XML for Long-term Preservation: Experiences from the DiVA Project* in Proceedings of the Sixth International Symposium on Electronic Theses and Dissertations, Berlin (2003). URL: http://edoc.hu-berlin.de/conferences/etd2003/hansson-peter/HTML/.

[3] Jacqueline Slats, *Practical experiences of the digital preservation testbed: Office formats* in Proceedings of the conference "File formats for preservation", Vienna (2004). URL: http://www.erpanet.org/events/2004/vienna/presentations/erpaTrainingVienna_Slats.pdf.

[4] Richard Anderson, Hannah Frost, Nancy Hoebelheinrich & Keith Johnson, "The AIHT at Stanford University: Automated Preservation Assessment of Heterogeneous Digital Collections", *D-Lib Magazine*, **11** (2005) URL: http://dlib.org/dlib/december05/johnson/12johnson.html.

[5] National Library of Australia, *Recovering and Converting Data from Manuscripts Collection Discs* (2002?). URL: http://www.nla.gov.au/preserve/digipres/recovering.html.

[6] Michael Lesk, *Preserving digital objects: Recurrent needs and challenges* in Proceedings of the 2nd NPO Conference on Multimedia Preservation, Brisbane (1995). URL: http://www.lesk.com/mlesk/auspres/aus.html.

[7] Wikipedia, *GIF* (2006). URL: http://en.wikipedia.org/wiki/GIF.

[8] Andreas Stanescu, "Assessing the Durability of Formats in a Digital Preservation Environment", *D-Lib Magazine*, **10** (2004) URL: http://dlib.org/dlib/november04/stanescu/11stanescu.html.

[9] Bruce D'Arcus, *Citations in "Open" XML* (2006). URL: http://netapps.muohio.edu/blogs/darcusb/darcusb/archives/2006/06/08/citations-in-open-xml.

[10] Microsoft, *Microsoft Office Open XML Formats Overview* (2005-6). URL: http://www.microsoft.com/office/preview/itpro/fileoverview.mspx.

[11] Microsoft Corporation, *Ecma International Standardization of OpenXML File Formats Frequently Asked Questions* (2006). URL: http://www.microsoft.com/office/preview/itpro/ecmafaq.mspx.

[12] Wikipedia, *GIF* (2006). URL: http://en.wikipedia.org/wiki/GIF.

[13] Wikipedia, *OpenDocument* (2006). URL: http://en.wikipedia.org/wiki/Open_document_format.

[14] OpenOffice.org, *Writer* (2006). URL: http://www.openoffice.org/product/writer.html.

[15] Steve Ball, *Multi-level Non-uniform Grouping of Very Large Flat Structured Documents* in Proceedings of AusWeb04, The Tenth Australian World Wide Web Conference (2004). URL: http://ausweb.scu.edu.au/aw04/papers/refereed/ball/paper.html.

[16] ERPANet, *ERPA Advisory* (2004). URL: http://www.erpanet.org/advisory/list.php.

[17] Microsoft, *Rich Text Format (RTF) Specification, version 1.8* (2004). URL: http://www.microsoft.com/downloads/details.aspx?familyid=ac57de32-17f0-4b46-9e4e-467ef9bc5540&displaylang=en.

[18] World-Wide Web Consortium, *Extensible Markup Language (XML) 1.0 (Third Edition)* (2004). URL:

http://www.w3.org/TR/REC-xml/.

[19] Library of Congress, *Sustainability of digital formats: XML* (2006). URL: http://www.digitalpreservation.gov/formats/fdd/fdd000075.shtml.

[20] Caroline Arms and Carl Fleischhauer, *Digital Formats: Factors for Sustainability,Functionality, and Quality* in IS&T Archiving Conference, Washington DC (2005). URL: http://memory.loc.gov/ammem/techdocs/digform/Formats_IST05_paper.pdf.

[21] World-Wide Web Consortium, *XSL Transformations (XSLT) Version 1.0* (1999). URL: http://www.w3.org/TR/xslt.

[22] Norm Walsh & Leonhard Muellner, *DocBook: The Definitive Guide*, O'Reilly (1999). URL: http://www.docbook.org/.

[23] OASIS Consortium, *OASIS* (1993-2006). URL: http://www.oasis-open.org/.

[24] Bob Stayton, *DocBook XSL: The Complete Guide*, Sagehill (2005). URL: http://www.sagehill.net/book-description.html.

[25] TEI Consortium, *The Text Encoding Initiative: Yesterday's information tomorrow* (2006). URL: http://www.tei-c.org/.

[26] Sebastian Rahtz, *XSL stylesheets for TEI XML* (2006). URL: http://www.tei-c.org/Stylesheets/teic/.

[27] The World-Wide Web Consortium, *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)* (2002). URL: http://www.w3.org/TR/xhtml1/.

[28] World-Wide Web Consortium, *Cascading Style Sheets, level 2CSS2 Specification* (1998). URL: http://www.w3.org/TR/REC-CSS2/.

[29] Apache, *The Apache Cocoon Project* (2006). URL: http://cocoon.apache.org/.

[30] Tim Bray, *Don't Invent XML Languages* (2006). URL: http://www.tbray.org/ongoing/When/200x/2006/01/08/No-New-XML-Languages.

[31] Peter Sefton, *OpenDocument or not you still need to Use Styles* (2005). URL: http://ptsefton.com/blog/2005/09/13/opendocument_or_not_you_still_need_to_use_styles..

[32] University of Southern Queensland, *Integrated Content Environment* (2006). URL: http://ice.usq.edu.au/.

[33] Leo Monus, Personal communication (2006).

[34] Tim Wilson-Brown, *xPub* (2004). URL: http://xml.anu.edu.au/xpub/.

[35] The University of Canberra, School of Creative Communication, *First*, (1995-2005). URL: http://www.canberra.edu.au/schools/creative-communication/events/anthology.

[36] Infinity Loop, *UpCast 5.4* (2006). URL: http://www.infinity-loop.de/products/upcast/.

[37] World-Wide Web Consortium, *Extensible Stylesheet Language (XSL) Version 1.0* (2001). URL: http://www.w3.org/TR/xsl/.

[38] RenderX, *XEP* (2006). URL: http://www.renderx.com/tools/xep.html.

[39] Peter Sefton, *PT's Outing: sustainable word processing* (2006). URL: http://ptsefton.com/blog.

[40] National Archives of Australia, *Xena software* (2005). URL: http://xena.sourceforge.net.

[41] National Archives of Australia, *An Approach to the Preservation of Digital Records* (2002). URL: http://www.naa.gov.au/recordkeeping/er/digital_preservation/summary.html.

[42] Ian Barnes & Scott Yeadon, *One-click DSpace Ingestion with the Digital Scholar's Workbench* in Proceedings of Open Repositories 2006, Sydney (2006). URL: http://www.apsr.edu.au/Open_Repositories_2006/barnes_yeadon.ppt.

[43] Ian Barnes, *Integrating the repository with academic workflow* in Proceedings of Open Repositories 2006, Sydney (2006). URL: http://www.apsr.edu.au/Open_Repositories_2006/ian_barnes.pdf.

[44] Ian Barnes & Peter Sefton, *Creating rich XML structure from word processing documents* (2006) In preparation.

[45] Peter Sefton, Personal communication (2005).

[46] Hans Liegmann, *Long-term preservation of electronic theses & dissertations* in Proceedings of the Sixth Interntational Symposium on Electronic Theses and Dissertations (2003). URL: http://edoc.hu-berlin.de/conferences/etd2003/liegmann-hans/HTML/liegmann.html.

[47] Open Source Technology Group, *SourceForge.net* (2001-2006). URL: http://sourceforge.net/.

[48] Apache, *Xalan* (2005). URL: http://xml.apache.org/xalan-j/.

[49] Apache, *Xerces* (2005). URL: http://xerces.apache.org/xerces-j/.

[50] Apache, *FOP* (2006). URL: http://xmlgraphics.apache.org/fop/.

[51] Ian Barnes, Christopher Blackall & Adrian Burton, *The digital scholar* (2006) In preparation.