The Digital Scholar's Workbench

Ian Barnes

Department of Computer Science, The Australian National University, Canberra, ACT Australia
Ian.Barnes@anu.edu.au

Abstract

In this paper I present the reasoning behind the development of a new end-to-end publishing system for academic writers. The story starts with investigating digital preservation of word processing documents. What file formats are suitable for long-term preservation of text? I believe that the answer is a high-quality structured XML format like DocBook XML or TEI. The next question is how do we get word processing documents into that format without incurring a prohibitive cost? Conversion is possible, but it requires human intervention at some point. It would be far too expensive to have archivists editing every document by hand on ingest, so how can we get authors to do the necessary work, particularly as most academics aren't at all interested in digital preservation of their work? The answer I propose is to offer them more than just an archiving solution. Instead of just getting preservation, they get a full end-to-end digital publishing solution, the digital scholar's workbench, tailored to their needs for document interoperability, collaboration and publication in multiple formats… oh, and they get preservation too.

Keywords: word processing; scholarly communication; digital preservation; XML; electronic publishing

1 Introduction

Word processing documents are a problem for digital repositories. They are not suitable for long-term storage, so they need to be converted into an archival format for preservation. This is not just a technical issue, but a people issue also. Most university repositories struggle to get academics to deposit their work: making the process more difficult will only make this problem worse. On the other hand, filling a repository with word processor documents that nobody will be able to read in a few years time, is a waste of time, effort and money. In this paper I will address the following questions:

1. What file formats are suitable for long-term storage of word processed text documents?
2. How can we convert documents into a suitable archival format?
3. How can we get authors to convert and deposit their work?

The answer I will propose is an end-to-end digital publishing system that allows authors to continue writing in a word processor, much as they do now, but provides an attractive range of file format conversion and publishing services: the Digital Scholar's Workbench.

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). In Section 2 I will briefly review some of the previous work in this area.

Most text documents created today are created in a word processor. In Section 3 I will discuss file formats, and give a tentative answer to Question 1 above. The file formats generated by word processors are generally not sustainable, so we need to consider converting documents to better formats. Many archives have chosen PDF, but this has serious problems. XML is a better answer, but it's not a complete answer as 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.

In Section 4 I will discuss possible methods available for converting word processing documents into a suitable XML format, addressing Question 2 above.

In Section 5 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.

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*. The DiVA people in Uppsala University Library are archiving documents in XML [1]. 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. At ANU we are considering something similar for large or complex documents, using RDF to describe the relationships between the parts. Slats [2] 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 [3] 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.

3 File Formats

3.1 Preservation Formats Versus 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 below is to store DocBook XML or TEI, but convert the document to 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.

3.2 Criteria for Sustainability

What features does a good preservation format have? How do we judge? Lesk [4] 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 satisfy this requirement.
3. *Open availability.* This means that proprietary formats are not acceptable. Remember what happened to GIF images when Unisys claimed that they were owed royalties because they own the file format [5]. What would happen if Adobe decided to do the same with PDF or Microsoft with Word?
4. *Interpretability.* It should be possible for a human to read the data, and also for small errors in storage or transmission to remain localised. This implies a strong preference for plain text rather than binary file formats. A small error in a compressed binary file can render the entire file useless.

Stanescu [6] looks at this topic from a risk management point of view. Slats [2] discusses criteria for choosing file formats, coming to very similar conclusions.

3.3 Word Processing Formats

Microsoft Word
The vast majority of all text documents created today are created in Microsoft Word using its native `.doc` format. 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 important reasons:

- Word format is owned by Microsoft corporation.
  - They could choose to change the format at any time, possibly forcing repositories to convert all their documents.

   - They could change the licensing at any time.
- Word format 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.
- Word is not just one format but many. Storing documents in Word format would force 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 and fixing any problems. Either way, this is an unacceptable cost.

Microsoft's new Office Open XML `.docx` format [7] is an improvement, but is still unsuitable for archiving. A `.docx` file is a compressed Zip archive of XML files. Compressed files are particularly prone to major loss if corrupted. Also some data is stored as strings that need parsing [8], rather than using XML elements or attributes to separate the different data items. This makes automated processing of these files much more difficult. Microsoft have released the Office Open XML specifications publicly, along with assurances that the format is and will always be free [9]. Despite the mistrust of many in the open source community, who remember the GIF/Unisys controversy [5], this appears to be genuine.

Open Document Format
Open Document Format (ODF) [10] is the native file format of OpenOffice.org Writer [11], the word processor component of the OpenOffice.org open source office suite. Open Document Format is an OASIS and ISO standard and a European Commission recommendation. It is also supported by KOffice and AbiWord.
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 means that the files are prone to catastrophic loss of content with even minor data corruption.

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 Office Open XML is still a proprietary format.
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 preserves all the formatting of most Word documents, with only minor differences. 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 [12], but at the moment only with documents that conform to a well-designed template. We are working to extend this to less carefully prepared documents, but the process is likely to require human supervision.

The other disadvantage of Open Document Format is that even for simple documents it is extremely complex. Unzipping a one-page document of about 120 words results in a collection of files totalling 300K in size. The formatting information is stored in a complex, indirect way. 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.

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 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.
- 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 [13]:

- The format is proprietary, owned by Adobe. While it is currently open, the company could decide to change this at any time.
- 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.
- PDF includes extra features like encryption, compression, digital rights management and embedding of objects from other software packages. These all present difficulties for archivists.

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. However, for the reasons given above, it is not a good preservation format.

3.5 RTF

RTF stands for Rich Text Format. It is a Microsoft specification [14], 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 proprietary, 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 [15]. 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, at least for relatively simple documents.

3.6 XML

XML [16] 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 [17] and the related conference paper by Arms & Fleischauer [18]. The main reasons are:

- 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 try to make the meaning apparent through the choice of element and attribute names).
- XML can easily be transformed into other formats using XSLT [19].

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. XML is only really useful 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.

DocBook XML
DocBook [20] 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 (particularly the O'Reilly books), but its application is wider, although it still seems a bit awkward and sometimes ill-matched to non-technical writing. DocBook is an OASIS [21] standard.

DocBook is huge, with over 300 elements. This makes it hard to learn, and cumbersome to use directly; few people create DocBook documents by hand. However this is 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 [22] for converting from DocBook XML into numerous formats including XSL-FO (and hence PDF) and XHTML. This is a huge headstart.

For converting word processor files to DocBook XML, 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.

TEI
TEI stands for the Text Encoding Initiative [23]. It is designed mostly for the preservation of literary and linguistic texts. 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 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.
- While it has a set of customisable XSLT stylesheets [24], the impression I get is that they are less mature and less comprehensive than the DocBook XSL stylesheets [22].

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).

XHTML+CSS
Since XHTML [25] is both valid XML and can be displayed by web browsers directly (with the formatting controlled by a CSS [26] stylesheet) this has been suggested as a possible archival format. I don't recommend it, for the following reasons:

- XHTML is 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 the section 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.

XHTML might be a good solution for low-value documents that archivists cannot afford to convert 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 [27].

Custom schemata
One of the biggest traps in the XML world is the idea that you create your own document schemata that perfectly match your particular needs. A university could create document types for lectures, lab exercises, reading lists, research papers, internal memos, minutes of meetings, rules, policies, agendas, monographs and so on. There are serious problems with this approach [28].

The first problem is maintenance. Each format will require converters to turn word processor documents into that format, and XSLT stylesheets for rendering into whatever viewing formats are needed: a reasonable short list would be HTML, PDF and plain text. What happens next is that someone wants to add an element to one of the document types. Every time this happens, you have to modify all the stylesheets for that document type. With multiple formats, there is likely to be demand for conversion between them: converting a stored research paper into a lecture, for example. The number of conversions needed grows fast.

The second problem is loss of 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 even more converters for exporting the documents in well-known interchange formats.

4 Converting Documents Into Archival Format

Having decided on a suitable archival format, the second issue is how to convert documents into that format. In a nutshell, this is solved by using OpenOffice.org to convert multiple formats (including Microsoft Word) into Open Document Format, then unzipping the result and applying one or more XSLT transformations to the pieces.

For some parts of the processing, particularly creating deep structure from a flat sequence of headings and paragraphs, XSLT can be quite cumbersome. Direct manipulation (for example using one of the various DOM bindings: Java, Python, Perl, PHP...) is an alternative worth considering.
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 easy 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 word processor 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 [29] is an example of this approach, as is the Digital Scholar's Workbench (see below). One of my current interests is in the possibility of creating a heuristic "structure guesser" that can use formatting information (indents and justification, space above and below, type size, weight and style etc) to make educated guesses about the structural roles of different paragraphs in documents that were not created using a good consistent set of styles. This is unlikely to ever be a perfect hands-off process however, and will probably always require some human supervision.

- 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). This would require trained staff, and costs time and money.
- 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 [30].

5 Usage

The main problem faced by institutional repositories (electronic archives) is no longer technical but social. Very few academics deposit their work. They're not interested, and it's too much trouble. After they finish preparing a piece of work for publication—sometimes a very time-consuming and frustrating process that can take days—they want to move on to the next piece of work. The last thing they want to do is start all over again reformatting and preparing their work for archiving, and then having to type lots of metadata for search and indexing.

I believe that the key to solving this problem is seeing archiving as just another form of publishing. Just like a journal, an archive has its technical requirements in terms of format, metadata and so on. Rather than having to go through this time-consuming and frustrating process by hand, more than once, it would be much better to create a system that can do the whole thing automatically or at least semi-automatically. This is what a good, end-to-end electronic publishing system should be able to do. Once we have a document and its associated metadata in a good, structured XML format, all this should be possible, and more including:

- Sending to a journal
- Submitting to a conference
- Depositing in an archive
- Posting to the department web site
- Posting to a personal blog
- Running off preprints to send to colleagues
- Adding the abstract to the department annual report
- Registering with research productivity measures

A system that offered all these services might be able to attract users from among the academic community. This is the challenge we are trying to meet with the Digital Scholar's Workbench. It's worth taking a few lines to discuss this in software engineering terms. What I'm proposing here looks very like "requirements creep", the process by which a simple software development project gets hopelessly bogged down by a constantly expanding list of requirements. Usually developers are instructed to get a list of requirements early in the process and then lock it down so that the size and complexity of the project can be contained.

In the case of this project, that would have defeated the purpose of the work entirely. A single-purpose piece of software that converts Word documents into DocBook XML already exists. (It is a commercial product called UpCast [31].) Apart from the expense involved in making it available across the university, the problem with this software is that very few ordinary authors will take the trouble to learn how to use it. This is because there is no incentive to do so. Academics don't care about preservation, so almost *any* effort is too much.

Some in the archiving community advocate taking an authoritarian stance and *requiring* academics to archive their work. Apart from any philosophical objections, this approach has another problem, that of quality of metadata attached to documents. If academics are forced to fill in lots of metadata forms, there will be a temptation to save time by entering rubbish. This defeats the whole purpose of archiving. If no-one can find your work, what is the point of preserving it?

Instead of forcing people to do something they don't want to do, the approach I support is to provide a tool that is so useful they will want to use it. Archiving comes for free as part of the package—once someone is using this tool, clicking a button to archive completed work is no trouble at all. Metadata can be scraped from the document, or at worst only has to be entered once when the document is created. Then it can be used and re-used whenever the document is published or archived or submitted to a journal or conference.

The point here is that what looks like requirements creep is actually a deliberate strategy to create something that will actually be used. By surrounding the archiving functionality with features academic writers will actually want, we make it far more likely that work will be deposited in the archive.

6 The Digital Scholar's Workbench

The Digital Scholar's Workbench [32, 33] is a prototype application that implements some of the ideas in this report. At the moment the Workbench is a web application that converts suitably structured word processing documents into archival quality DocBook XML, and then from there into XHTML for onscreen viewing and into PDF for printing.

In order to work with the current version of the workbench, documents must be written using the USQ ICE template [29]. This template has a single set of all-purpose styles [34] designed so that it is possible to automate the conversion to DocBook XML. 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 [35], 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, who states that, "All of us … have experienced that you need to tolerate and to adhere to a structured framework in order to profit from its advantages." [36] At the time of writing, the Digital Scholar's Workbench has the basic functionality of being able to convert word processing documents written using the USQ ICE template into DocBook XML and from there into XHTML and PDF. Further development will focus on:

- making its support for Word and Writer documents more robust, and supporting more of the features, available in the word processing software;
- improving the links to repository software, so that documents can be deposited, together with associated metadata and any linked images or other resources, with one click;
- adding one-click publishing to blogs, websites and learning management systems;
- reformatting papers ready for submission to journals and conferences, via a plugin mechanism, so that once an export plugin has been created it can be contributed back to the community for use by others;
- support for complex multi-part documents like books and theses;
- articulation with desktop publishing software to produce high-quality typeset PDF output;
- improved metadata entry and storage;
- linking with a version control system, so that authors have access to all previous versions of their documents at all times;
- round-tripping of documents back to Word or Open Document Format to enable seamless collaboration between co-authors using different word processing software;
- platform- and software-independent bibliography management (perhaps outside the limited and difficult-to-use systems built in to Word and Writer);
- adding support for TeX and LaTeX documents;
- support for presentation slides; and
- attempting to remove the requirement that documents be written using only the set of styles in the ICE template.

The prototype workbench will eventually be made available to developers and early adopters as an open source software project, probably through SourceForge [37].
The Digital Scholar's Workbench is built on open-source technology. The current version uses the Apache Cocoon [27] web application framework, which incorporates the Xalan XML parser [38], the Xerces XSLT processor [39] and the FOP XSL-FO processor [40]. It also uses OpenOffice.org in headless mode to transform Word documents into Open Document Format. It relies on the USQ ICE template. (This architecture may change over the coming months.)

7 Example/Colophon

This document was begun in OpenOffice.org Writer on Linux, using the template provided by the conference organisers. When I couldn't format the reference list correctly using the bibliographic support built in to Writer, I moved the document to Microsoft Word on my Mac, and used EndNote [41] for the bibliography formatting. At first this didn't work; something Writer had done to the document meant that Word and EndNote couldn't talk to each other. In order to fix this, I had to start again with a fresh copy of the conference template and cut and paste my content across. Within the time constraints, there appeared to be no practical way to extract my bibliographic data from Writer and import it into EndNote, so I had to enter 43 references by hand. I estimate that I spent at least one entire working day simply wrestling with my word processing and bibliographic software, rather than working on the actual content of the paper.

Imagine now that the Digital Scholar's Workbench existed as described above. None of this would have been a problem. Transferring the paper from Writer to Word would have been accomplished via the DocBook interchange format. I'm not sure how the bibliography support will work—this may well be the hardest part of the work planned, or it may be that a service like Zotero [42] can do everything required—but there would certainly be no need to rekey entries from a bibliographic database. With an appropriate output filter, the workbench could have reformatted my paper to conform to the conference template, saving me the trouble. Alternatively, the conference organisers could have accepted the paper in DocBook XML, giving them the flexibility to typeset the proceedings for publication, and turn them into a web site.

8 Conclusion

This project was originally about digital preservation only. The Australian Partnership for Sustainable Repositories wanted to know how to preserve word processing documents. Answering that question led to the need to convert documents into structured XML for preservation. From there, the question was not just "How?", but also "How will we get people to actually do this?" The prototype Digital Scholar's Workbench is an attempt to answer both questions. The first is a purely technical question, but the second one is a people question, in the realms of the sociology of knowledge production. There is probably a long way to go with this, and the form of the software will change as we get feedback from focus groups of academics. The principle is that there is little chance of getting our users to do what the university archive wants them to do—convert their work into structured XML and deposit it in the archive—without offering them something they want, namely vastly simplified workflows for their everyday writing and publishing tasks.

Acknowledgements

This work was funded by the Australian Commonwealth Department of Education, Science & Training, through the Australian Partnership for Sustainable Repositories (APSR), which is part of the government's Systemic Infrastructure Initiative, part of "Backing Australia's Ability—An Innovative Action Plan for the Future". I thank the Australian National Unversity'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 and Margaret Henty. I also thank the ANU's Department of Computer Science for releasing me from teaching while I did this work. I also thank Peter Sefton and his team at the University of Southern Queensland for many useful discussions particularly relating to the USQ Integrated Content Management system (ICE) which has a lot in common with the Digital Scholar's Workbench. I also thank them for creating the ICE word processor templates which I use in the current version of the Digital Scholar's Workbench. This paper is based on, and extends, an earlier report written for and published by APSR [43].

Notes and References

[1] MÜLLER, E.; KLOSA, U.; HANSSON, P.; ANDERSSON, S.; SIIRA, E. Using XML for long-term preservation: Experiences from the DiVA project. Sixth International Symposium on Electronic Theses and Dissertations. Berlin, 2003. URL: http://edoc.hu-berlin.de/conferences/etd2003/hansson-peter/HTML/

[2] SLATS, J. Practical experiences of the digital preservation testbed: Office formats. File formats for preservation. Vienna, 2004. URL: http://www.erpanet.org/events/2004/vienna/presentations/erpaTrainingVienna_Slats.pdf

[3] ANDERSON, R.; FROST, H.; HOEBELHEINRICH, N.; JOHNSON, K. The AIHT at Stanford University: Automated preservation assessment of heterogeneous digital collections. D-Lib Magazine 2005;11. URL: http://dlib.org/dlib/december05/johnson/12johnson.html

[4] LESK, M. Preserving digital objects: Recurrent needs and challenges. 2nd NPO Conference on Multimedia Preservation. Brisbane, 1995. URL: http://www.lesk.com/mlesk/auspres/aus.html

[5] GIF. Wikipedia, 2006. URL: http://en.wikipedia.org/wiki/GIF

[6] STANESCU, A. Assessing the durability of formats in a digital preservation environment. D-Lib Magazine 2004;10. URL: http://dlib.org/dlib/november04/stanescu/11stanescu.html

[7] Microsoft Office Open XML formats overview. Microsoft, 2005-6. URL: http://www.microsoft.com/office/preview/itpro/fileoverview.mspx

[8] D'ARCUS, B. Citations in "Open" XML. 2006. URL: http://netapps.muohio.edu/blogs/darcusb/darcusb/archives/2006/06/08/citations-in-open-xml

[9] Ecma international standardization of OpenXML file formats frequently asked questions. Microsoft, 2006. URL: http://www.microsoft.com/office/preview/itpro/ecmafaq.mspx

[10] OpenDocument. Wikipedia, 2006. URL: http://en.wikipedia.org/wiki/Open_document_format

[11] Writer. OpenOffice.org, 2006. URL: http://www.openoffice.org/product/writer.html

[12] BALL, S. Multi-level non-uniform grouping of very large flat structured documents. AusWeb04, The Tenth Australian World Wide Web Conference, 2004. URL: http://ausweb.scu.edu.au/aw04/papers/refereed/ball/paper.html

[13] ERPA Advisory. ERPANet, 2004. URL: http://www.erpanet.org/advisory/list.php

[14] Rich Text Format (RTF) Specification, Version 1.8. Microsoft, 2004.

[15] Recovering and converting data from manuscripts collection discs. National Library of Australia, 2002. URL: http://www.nla.gov.au/preserve/digipres/recovering.html

[16] Extensible Markup Language (XML) 1.0 (Third Edition). World-Wide Web Consortium, 2004. URL: http://www.w3.org/TR/REC-xml/

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

[18] ARMS, C.; FLEISCHHAUER, C. Digital formats: Factors for sustainability, functionality and quality. IS&T Archiving Conference. Washington DC, 2005. URL: http://memory.loc.gov/ammem/techdocs/digform/Formats_IST05_paper.pdf

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

[20] WALSH, N.; MUELLNER, L. DocBook: The definitive guide. O'Reilly, 1999. URL: http://www.docbook.org/

[21] OASIS. OASIS Consortium, 1993-2006. URL: http://www.oasis-open.org/

[22] STAYTON, B. DocBook XSL: The complete guide. Sagehill, 2005. URL: http://www.sagehill.net/book-description.html

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

[24] RAHTZ, S. XSL Stylesheets for TEI XML. 2006. URL: http://www.tei-c.org/Stylesheets/teic/

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

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

[27] The Apache Cocoon Project. Apache, 2006. URL: http://cocoon.apache.org/

[28] BRAY, T. Don't invent XML languages. 2006. URL: http://www.tbray.org/ongoing/When/200x/2006/01/08/No-New-XML-Languages

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

[30] MONUS, L. Personal communication. 2006.

[31] upCast. Infinity Loop, 2003-2007. URL: http://www.infinity-loop.de/products/upcast/

[32] BARNES, I.; YEADON, S. One-click DSpace ingestion with the Digital Scholar's Workbench. Open Repositories 2006. Sydney, 2006. URL: http://www.apsr.edu.au/Open_Repositories_2006/barnes_yeadon.ppt

[33] BARNES, I. Integrating the repository with academic workflow. Open Repositories 2006. Sydney, 2006. URL: http://www.apsr.edu.au/Open_Repositories_2006/ian_barnes.pdf

[34] SEFTON, P. 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

[35] SEFTON, P. Personal communication. 2005.

[36] LIEGMANN, H. Long-term preservation of electronic theses & dissertations. Sixth International Symposium on Electronic Theses and Dissertations. Berlin, 2003. URL: http://edoc.hu-berlin.de/conferences/etd2003/liegmann-hans/HTML/liegmann.html

[37] SouceForge. Open Source Technology Group, 2001-2006. URL: http://sourceforge.net/

[38] Xalan. Apache, 2005. URL: http://xml.apache.org/xalan-j/

[39] Xerces. Apache, 2005. URL: http://xerces.apache.org/xerces-j/

[40] FOP (Formatting Object Processor). Apache, 2006. URL: http://xmlgraphics.apache.org/fop/

[41] EndNote 9. Thomson ResearchSoft, 2005. URL: http://www.endnote.com/

[42] Zotero. Center for History and New Media at George Mason University, 2006. URL: http://www.zotero.org/

[43] BARNES, I. Preservation of word processing documents. Australian Partnership for Sustainable Repositories, 2006. URL: http://www.apsr.edu.au/publications/preservation_of_word_processing_documents.html