HTML5 for ETDs

Virginia Polytechnic Institute and State University

CS 4624

May 8th, 2010

Sung Hee Park

Dr. Edward Fox

Nicholas Lynberg

Philip McElmurray

Jesse Racer

Table of Contents

Executive Summary

This project aims to develop software and to demonstrate that software's utility, with the goal of reformatting plain ETDs (Electronic Theses and Dissertations) into HTML5. This can be thought of as a migration to a better format for preservation. Most current ETDs are submitted in PDF (Portable Data format); some are accompanied by multimedia files or other files connected with hypertext links. Some have links to multimedia files and so are hypermedia works. HTML5 allows a single file to encode multiple media types and support linking among those. For readers of ETDs it would be much more convenient to work with a single file that has all of the content linked together, including all of the multimedia information in the ETD; thus our goal is to transform to HTML5, to aid browsing of ETDs.

Virginia Tech has a digital library for ETDs. Through its search service, the VT-ETD digital library provides the metadata (title, author, abstract, date, etc.) for the ETD, plus the ETD itself in PDF file format. In addition, the metadata has links, for those ETDs that have additional multimedia files (like, avi, mpeg, wav, mp3, gif, etc.), to each of those files. But, after we find the ETD that we are looking for, it is inconvenient to view the ETD in PDF, and then to display each of the separate multimedia files. Users who view an ETD with multimedia files must use one viewer for PDF and another for each type of multimedia content. For user's viewing convenience, an ETD digital library should support HTML5. HTML5 facilitates viewing multimedia by integrating multimedia with video and audio tagging.

User's Manual

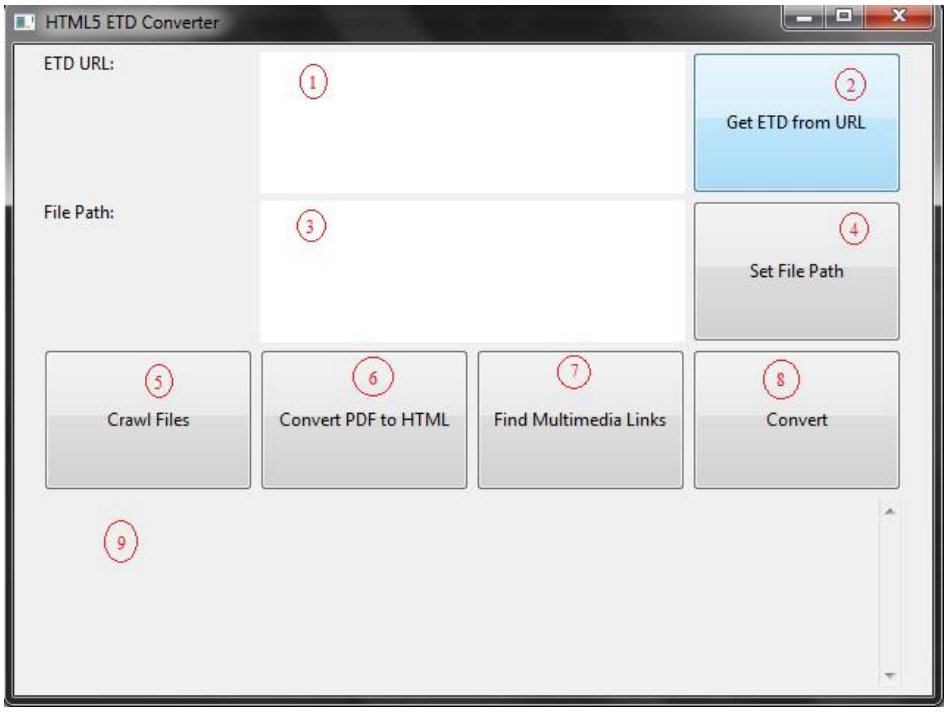

Figure 1 – User Interface for migration.

1. Enter a URL of the ETD and multimedia files. For example:
   http://scholar.lib.vt.edu/theses/available/etd-06152006-024611/
   Do not enter the location of the PDF file
2. Click the "Get ETD from URL" button to retrieve the ETD.
3. Enter a file name for the finished etd (e.g. "etd.html").
4. Click the "Set File Path" button.
5. Click the "Crawl Files" button. If Perl is not installed on the machine the converter is running on the crawler will not work.
6. Click the "Convert PDF to HTML" button.
7. Click the "Find Multimedia Links" button. Currently this does not work due to issues with the JAVA runtime execution command. The links can be tagged by running the included perl script with the syntax "perl link_tagger.pl [output file for crawl] [etd file]" from a Windows command line.
8. Click the "Convert" button.
9. Output box to show the progress of your conversion.

The entire conversion process may take a while. If there are video files that need to be converted, they will take time. We have converter software that converts at a one to one ratio for these video files. For the text and image extraction with a document of roughly 230 pages and with 232 images found (with

the problems noted later about image processing), it takes roughly 20 seconds for the text extraction and another minute for the image extraction. For the image extraction, this does not include inserting the images back into the HTML file.

Developer's Manual

Concept Map

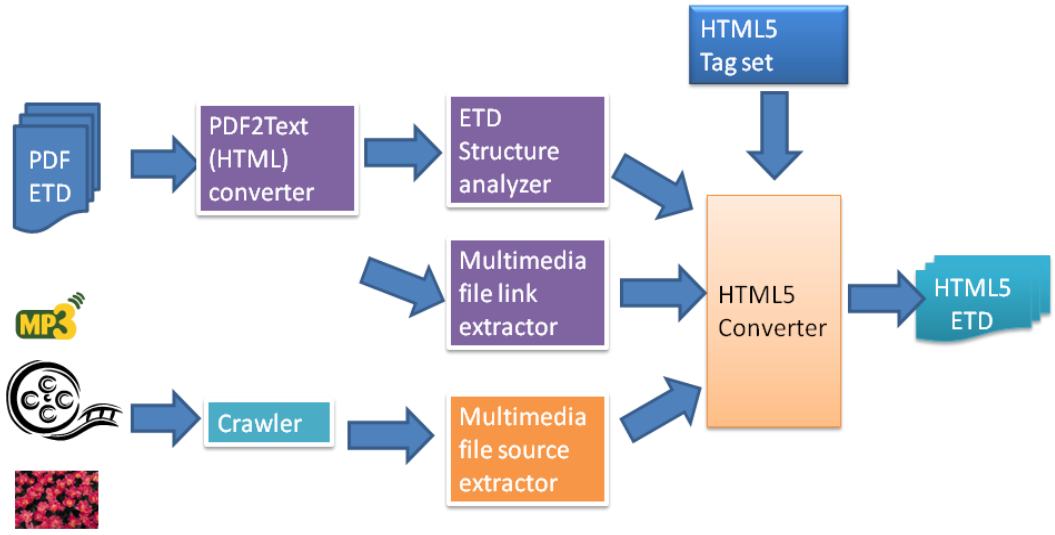

Figure 2 – Concept Map

The crawler retrieves all of the media files, if they exist, and have them saved. Based on current HTML5 constraints, we must download any file that is an avi file because the video tag does not support avi files. We need to convert them into ogv files so that Mozilla Firefox and Google Chrome can actually view them. Currently, the multimedia files, the video files, for the ETDs are stored as avi. This is one problem that we have come across. There is an embed tag that HTML5 has but we ruled this out as a good use because we didn't want to use a tag that would depend on the user's computer to have installed software to view the video.

Next the PDF ETD is converted to text and then to HTML. This will consist of a Java program that uses PDFBox to convert the PDF into text. PDFBox is an open source Java PDF library for working with PDF documents. The library allows creation of new PDF documents, manipulation of existing documents and the ability to extract content from documents. Since there is no need to create new PDF documents, we will be working mainly with the manipulation portion to extract the text and images, if they exist. [1]

The conversion process to text consists of the PDF being read in by PDFTextStripper, which converts it all to a string, and then we output this string into an HTML file which contains just the text from the PDF. There are also some images that are embedded in the PDF that need to be extracted. PDXObjectImage is the main object type we used to access images. We came across a problem with using this. If the image used when creating the PDF was a jpg file, then most of the images comes back complete; with almost all png files, the image comes back in parts and doesn't contain the entire image in one file.

Inventory of Files

The source files we have for the project are:

ManualConvert.java – extracts the text and images from the PDF.

MultimediaCrawler.java – Uses perl script to crawl multimedia files

SWTDriver.java – creates the UI that runs all the background code to do the migration

Media_crawler.pl – perl script to crawl for media

link_tagger.pl – perl script to tag media in etd

Media_tagger.pl – perl script to tag media

There are also many libraries that are included in the project, these are necessary for the program to run.

Lessons Learned

Timeline/Schedule

2/12 HTML5 study

2/19 tag set selection for HTML5ETD

2/ 26 HTML5ETD manual construction

3/ 5 prototype crawling, multimedia link location finding

3/ 17 HTML5 tag conversion ETD

3/18 midterm presentation

3/26 HTML5 tag conversion ETD2

4/9 HTML5 tag conversion ETD3

4/16 refined crawling, multimedia link location finding

4/23 HTML5 ETD application (mobile)

4/29 final presentation

Problems

The first problem we faced when given this project was how to go about migrating from the PDF file into a HTML5 file. First off, we wanted to extract all of the text from the PDF file. This was a problem at first, there didn't exist a standard Java library that we could use to extract text from the PDF file.

The second problem we faced was that in some of the PDF files we had, there were images that existed. Obviously a text extraction tool would not allow for us to extract the images out of the PDF.

After committing to a library, PDFBox, to extract all our text and images we ran into a problem with the formatting and styling of the text. When the text is taken out of the PDF, it is plain text and no longer keeps the size, color, ect of the font that was being used. This is bad because we lose our headings and such in the HTML file. Also, some of the images when in the PDF appear to be in many parts stacked on top of one another. When they are extracted, many pieces of one image are found.

HTML5 itself is a new technology. It is the next major revision of HTML. The main goal of HTML5 is to reduce the need for proprietary plug in based rich internet applications. [3] Because of this, not all

browsers support full HTML5 capabilities. One example of this is the video tag. The latest versions of Internet Explorer and Opera do not support the video tag at all but Mozilla Firefox, Google Chrome, and Safari all support it. Also, we may need to deal with audio files, where Firefox, Chrome and Safari all support it but IE and Opera do not.

There is also a great need for HTML5 to be viewed on mobile devices, such as cell phones or iPods. To date, the only thing that supports this is the iPod touch, iPhone, and iPad. Cell phones with Android 2.1 and Blackberry's don't support HTML5 video. This was a main concern for our project because of the amount of people that use mobile devices every day.

Solutions

PDFBox was the best solution we found to solving the problem of dealing with extracting the text and images from the PDF. There exist classes in PDFBox to easily take out the text and images from the PDF and put the text into a string and this string is outputted between body and pre tags. The images are saved in the source path images file. As for the problem with multiple parts for one image, we have no real solution to this problem. It seems to be something to do with the image type when the PDF was created.

For the problem with the browser type, it will be assumed that if the user wants to view the video and/or audio elements then they should use a browser which supports it. HTML5 also has an embed tag. The embed tag only allows for a standalone media player, i.e. Windows Media Player, Flash, ect, to play the video. We wanted to get away from relying on the user's computer to have the appropriate software, we are not going to use the embed tag to view video.

As for the problem file types that can be used with the video tag, we propose converting all files to ogv. This will allow for the most browser types to view the videos. We have converter software that will convert to ogv; the only downside to this is that we must download the video, convert it and then put it into the file. We are suggesting a better solution than this to the problem. The files that are not ogv already should be converted to ogv before putting them onto the website. This would speed up the process because instead of downloading and converting, which can take a lot of time, we would be able to just have it point to where the file is and not actually have it downloaded again.

Acknowledgements

The project sponsor:

Sung Hee Park

2030 Torgersen Hall

shpark@vt.edu

Dr. Edward Fox

2160G Torgersen Hall

fox@vt.edu

References

[1] *Apache PDFBox - Apache PDFBox - Java PDF Library*. Web. 07 May 2010. <http://pdfbox.apache.org/index.html>.

[2] *HTML5 Demos and Examples*. Web. 07 May 2010. <http://html5demos.com/>.

[3] "HTML5." *Wikipedia, the Free Encyclopedia*. Web. 07 May 2010. <http://en.wikipedia.org/wiki/Html5>.