Quantifying Software Requirements for Supporting Archived Office Documents using Emulation

Thomas Reichherzer and Geoffrey Brown
Computer Science Department
Indiana University, Lindley Hall 215
Bloomington, IN 47405, U.S.A.
{treichhe, geobrown}@cs.indiana.edu

ABSTRACT

This paper addresses the issues associated with building software images to support a collection of archived documents using machine emulators. Emulation has been proposed as a strategy for preservation of digital documents that require their original software for access. The creation of software images is a critical component in archiving documents via emulation. The software images include the operating system, application software, and supporting software artifacts such as fonts and Codecs (Compression-Decompression algorithm). A practical emulation environment to support a digital document requires both an emulator and a software image. This paper considers the issues associated with creating such software images to support Microsoft Office documents. In particular, we discuss a set of software tools and strategies that we developed to automatically analyze the dependencies of Microsoft Office documents on software resources and supporting files. As a proof of concept, the tools and strategies have been applied to establish dependencies of Office documents from a document library containing approximately 200,000 documents and to automatically collect missing resources such as fonts. The software tools are a first step toward an interactive system that aids in the construction of robust emulation environments for preserving digital artifacts. However, they may also be used in other contexts, for example, to support screening of documents for archiving and migration to new platforms to ensure correct visualization.

Categories and Subject Descriptors

H.3.7 [**Information Storage and Retrieval**]: Digital Libraries—*Collection, Systems issues*

General Terms

Documentation, Experimentation

Keywords

digital preservation, emulation, Office documents

JCDL'06, June 11–15, 2006, Chapel Hill, North Carolina, USA.

1. INTRODUCTION

Emulation has been widely discussed as a preservation strategy for digital artifacts such as multimedia presentations that are intimately tied to their original hardware and software platform for interpretation [15, 12, 13]. The basic idea of emulation is simple - in order to preserve a complex digital artifact in the face of rapid hardware obsolescence, an emulation platform is created consisting of a program to simulate the original hardware platform (the emulator) and the original software used to access the artifact (the software image). Future access to the artifact is achieved by executing the emulation platform on modern hardware in place of the original platform. Emulation has been successfully tested to preserve individual artifacts such as the BBC Doomsday book project and various multimedia artworks [16], but it has not been tested as a means for preserving a large collection of digital artifacts.

There is a fundamental difference between preserving individual digital artifacts and a large collection of artifacts expressed by two tasks that must be addressed to build reliable emulation platforms. First, the software (hardware) resource requirements must be established and second, the emulation platform must be tested to ensure that it adequately supports the artifacts. In the case of individual artifacts, each of these tasks can be accomplished with significant human intervention assuming that the artifact is of sufficient cultural value to justify building a specialized emulation platform. When archiving large collections, it is often difficult to predict which, if any, of the artifacts will be of value in the future. Therefore, it is generally infeasible to spend significant resources on any individual artifact. For example, PowerPoint presentations may differ significantly in the fonts, language support, video and audio Codecs (Compression-Decompression algorithm), and "helper applications" required for successful access. When creating an emulation platform to support a collection of artifacts, each artifact must be evaluated to determine its software and hardware resource requirements. Once an emulation platform has been implemented, each artifact should be tested against the platform to ensure that at least basic functionality is achieved, and a careful quality assurance program must be followed to ensure that loss of artifact specific functionality is minimized. A further complication is that a collection of artifacts may have conflicting software requirements and hence a supporting emulation platform may require a multitude of software images to support the collection.

The primary focus of this paper is to demonstrate automatic analysis of software requirements for a large collection of documents and to quantify these requirements for Office documents. In addition, we describe first steps in developing tools to test software images supporting a document collection. All software tools, presented in this paper, have been primarily developed to perform automatic document analysis and image testing for the purpose of building robust emulation environments; yet, we recognize their general usefulness in other contexts. For example, the tools may be used to screen documents for requirements before accepting them into an archive that imposes certain restrictions on fonts or embedded multimedia resources. The tools may also be used to support migration of documents to different platforms that may have different resource requirements.

Automatic analysis of software requirements for Office documents is limited by the information that the document's object model provides. An experimental study on a large collection of Office documents has shown the limitation of the information that can be reliably extracted. For example, while links to embedded audio and video files can be reliable identified, information about the Codec needed to play back a media file cannot be determined from the object model. Only by accessing the media file itself can Codec requirements be established. Similar problems occurred when establishing the font requirements. While a font's face name can be extracted from a document object model, the object model provides no specific information to uniquely identify a font used in a document. The face name itself is generally not sufficient to identify a font because face names are user defined and currently lack any standardization. As a consequence, two fonts may be identical, even though their face names may be different. Examples of such cases have occurred in our experiment and are discussed in section 4. Other issues, where the object model provided insufficient information are also discussed in the experimental section.

The results from the experimental study indicate that authors of Office documents use a large variety of fonts that differ from the set of fonts provided by Microsoft Office and the Windows operating system. We believe that fonts not supplied by Microsoft must come from software applications installed on the author's platform or from font vendors consulted by the author to add additional fonts to their platform. Another major result from the study is that the number of Codecs required to play back media files embedded in Office documents is small. This makes it feasible to manually collect the necessary Codecs when building emulation platforms to guarantee proper playback of audio and video files. Finally, the study has also shown that any embedded objects, other than movie and audio clips, are always correctly visualized by Office even if the application with which the embedded object was created is not installed.

We have chosen to focus upon the issues related to creating and testing software images rather than the creation of emulators. There are several emulators and virtual machines available for the ubiquitous x86-based PC. Thus, we believe the creation of reliable hardware emulator is largely a solved problem. For example, VirtualPC [18] and QEmu [7] provide reliable platforms to execute Windows 3.1 through XP operating systems on Windows, Linux, and Macintosh computers. VMWare [17] provides robust virtual PC emulators on Linux and Windows. The primary difference between emulation and virtualization is whether instruction set simulation is required. Instruction set simulation is relatively easy. For example, GDB (a popular debugger) includes instruction set simulators for most common microprocessors [20].

The rest of the paper is organized as follows. Section 2 proposes an architecture for preserving digital artifacts and discusses some of the challenges for building robust emulation environments that support access of Microsoft Office documents. Section 3 discusses first how to analyze Office documents before presenting a suit of software tools and strategies to establish platform requirements. Section 4 describes an experimental study to demonstrate how the task of emulation platform specification can be accomplished when faced with large collections of digital artifacts. Issues that emerged from the experimental study are discussed in section 6. Finally, section 7 concludes our work.

2. AN ARCHITECTURE FOR PRESERVING DIGITAL ARTIFACTS

We envision a digital archive utilizing emulation to consist of repositories for holding artifacts to be viewed, software to generate emulation images on-demand for viewing individual artifacts, and compute servers for executing the emulation environment. Figure 1 illustrates this architecture. If a user requests an artifact, a query is generated corresponding to the artifact requirements in order to retrieve the necessary operating system, application software, and supporting files. These components are used to generate an emulator image that is executed on the compute server to generate a response. In this case, a response consists of an interactive session for viewing the artifact. The resulting multimedia display appears on the user's workstation. In the remainder of this section, we address the issues associated with building robust emulation platforms to support this model. The primary technical issues are analysis of document requirements, collection of necessary support components, generation of emulation images and testing of generated images.

To ensure that the emulation environment successfully supports viewing and access of the various artifacts, the requirements must be known and stored with the artifacts in the Artifact Repository. In addition, software and supporting files such as fonts must be collected and stored in the Software Resource Repository for subsequent access to dynamically build a suitable emulator environment. The creation of such repositories must be done far in advance of any users accessing the digital artifacts to view them on a workstation. Since technical obsolescence may make it impossible or impractical to "fix" an emulation environment when a fault is discovered by a user accessing a particular artifact, the process of creating the emulation environment must provide high confidence that such failures are unlikely. Building emulation environments must scale with the requirements of an archive storing a diverse, large number of artifacts. With tens of thousands of artifacts, it is not feasible for human beings to be required to "touch" each artifact in creating and testing emulation environments. Thus, an automatic approach is needed to solve this problem.

To create an emulation environment, the software and operating system resources required to access artifacts must be established. Assuming the requirements are known, creating an emulation "image" consists of installing an operating sys-

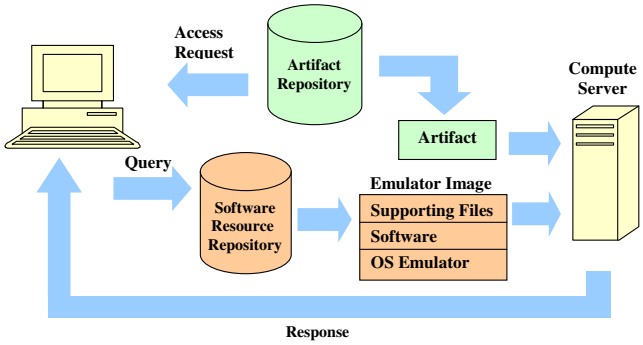

Figure 1: An Architecture for Viewing Digital Artifacts.

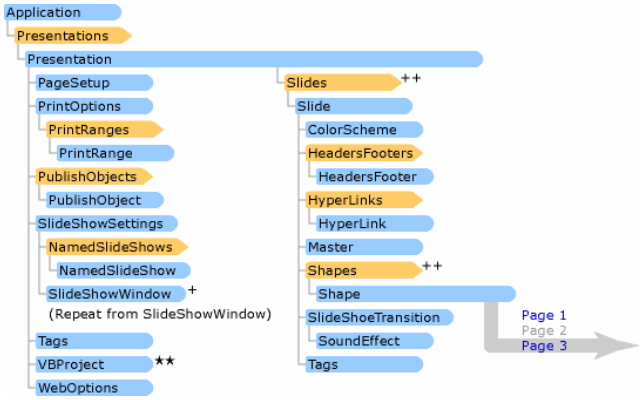

Figure 2: A section of the PowerPoint object model.

tem emulator for a target computing platform, applications and plug-ins, and additional resources. The image can be executed to view artifacts or stored in the Software Resource Repository for later use. Multiple images may be generated and stored to support sometimes conflicting software requirements of artifacts. Once created, emulation images can be customized at any time to meet the visualization requirements for new artifacts by adding executables or other supporting files to the image. We envision that when such customized images are booted for the first time to visualize an artifact, the compute server will install first the necessary patches and applications before generating a response.

Digital artifacts, including Microsoft Office documents, commonly link to other artifacts with different formats. This means that archival preservation of a given artifact may involve the preservation of additional artifacts of different formats with different software and resource needs. The situation is further complicated because the process of collecting artifacts into an archive may result in breaking many of the links between them which have location specific references. The linking problem could be solved in an emulation environment through the use of "proxies" which redirect references to linked artifacts to their appropriate location in an archive and either launch helper applications in the emulation environment of the source artifact or launch separate emulators to view the linked artifacts. Thus, emulation support for an artifact requires both the emulator and a proxy environment for the parent archive. While this paper does not address directly the issue of handling linked artifacts, it will present solutions for capturing the requirements of linked artifacts to ensure that the Office documents are rendered correctly. For example, for linked artifacts such as video and audio clips, requirements to properly play back such linked clips include media players that can handle the file format and the Codecs.

In conclusion, to build robust emulation environments that support visualization of a large collection of documents, three major tasks must be performed. First, the digital artifacts from the artifact collection must be analyzed to establish their software requirements for correct visualization. Second, emulator images must be generated that includes the operating system, application software and supporting files to visualize an artifact when access is requested. Third, the image must be tested to ensure that future attempts to access and interact with a digital artifact do not fail due to inadequacies in the emulator platform. In this paper, we focus primarily on the first task and describe first steps toward accomplishing the third task. The second, the creation of a test image, is subject of future research.

3. DOCUMENT ANALYSIS

This section discusses first how document analysis is automatically performed using existing Microsoft programming interfaces. It then presents the software tools that were recently developed to establish resource requirements for a large collection of Office documents.

Microsoft applications including Word and PowerPoint provide a programmer accessible interface that allows programs to access and control a document's object model. This technology, known as COM (Component Object Model) and the OLE (Object Linking and Embedding) [8] Automation programming interface, provides the basic interface to read an Office document, parse the document's object model, and extract the information necessary to determine resource requirements. For example, to determine the fonts used in a PowerPoint document, a program running on a platform where the Office software is installed can automatically load the PowerPoint document into memory, traverse the object model of a presentation [6], depcited in Figure 2, loop through the presentation slides and the characters on each slide, access the Font object and determine the font's face name. Similarly, embedded objects such as audio and video clips and their corresponding file formats can be established and the resource links to the objects used to access them for further analysis. The pseudo-code in Table 1 illustrates how the object model for PowerPoint presentations were parsed. Only slight modifications were needed to apply the same algorithm for parsing Word documents.

The OLE Automation programming interface not only allows access to an Office document's object model, but facilitates control of the model to automate document navigation and modification. This can be useful to build interactive systems that aid in resolving detected visualization problems in documents. Although automatic problem resolution is desirable, especially if there are a large number of documents that need to be preserved permanently, it is not realistic to expect that every visualization problem in a document can be resolved automatically. Human inspection is necessary and, assuming difficult to resolve problems occur

```
PROCEDURE Parse Object Model
INPUT:
  PPT: a PowerPoint presentation
OUTPUT:
  A record summarizing document attributes, font
  requirements, linked resources, and embedded objects
BEGIN
  Access Document Properties DP in PPT
  extract and print attributes and their values from DP
  For each slide S in PPT do
    For each shape C in S do
      If C is a text frame, do
       extract text range R
       for each character in R, extract font information
       store font information, if new
      End If
      If C is a linked object, do
       extract media type // i.e. audio or video
       extract file link
       store file link and media type, if new
      End If
      If C is embedded object, do
       extract program ID and store it if new
      End If
    End For
  End For
  display font information
  display information about linked audio/video clips
  display program IDs of embedded objects
END
```

Table 1: Pseudo-code for parsing the PowerPoint object model.

rarely, requesting an inspector to examine problem areas is a viable approach to ensure that (1) artifacts are complete when added to an artifact repository and (2) emulation environments are robust because the necessary applications and resources are available to guarantee future access of a set of stored artifacts.

Despite the fact that useful information can be extracted from a Microsoft Office object model, there are serious limitations affecting the process of automatically establishing resource requirements. For example, no class in the object model provides version information about the Microsoft Office application with which a document was originally created. Only the name of the application can be extracted but this name does not reliably indicate the version of the Office software or the platform on which the Office software run at the time the document was authored. Another limitation is the information that is available about fonts. Class *Font* in the object model provides information about a font's face name, style, size, color, and effects but it does not provide a kind of font "fingerprint" that would allow a font to be uniquely identified. As a result, fonts can only be distinguished on the basis of their face name, which is not reliable. For example, in the experiment discussed below, we determined that the word "times" as in Times New Roman, a commonly used font, occurs 146 times in distinct face names of fonts extracted from PowerPoint presentations. How many of these fonts actually refer to the font Times or Times New Roman cannot be determined. Finally, as was previously mentioned, the Office object model does not provide media type or Codec information of embedded media files. This problem, however, can be addressed by analyzing the media file directly. Software tools for analyzing such media files as well as Office documents will be discussed next.

Software Tools for Automatic Artifact Requirement Analysis

A suit of software tools was developed to perform a requirement analysis for Office documents and associated artifacts. They include C++ programs to parse the Office object models and to identify installed fonts on a computing platform, Python scripts to compile a list of resource requirements for Office documents and to analyze and compare fonts, and Java programs to search and download Office documents and fonts. In addition, off-the-shelf freeware products were used to analyze media files and extract font attributes from font files. The tools proved to be sufficiently robust and fast enough to process large document libraries. Difficulties occurred when Office documents were corrupt, access was restricted, or user interaction was required to complete file processing. Not everything could be fully automized. For example, to extract Codec information from media files, interaction with a graphical user interface was required to select media files to be processed and to store the processing results in a text file.

The C++ programs that perform document analysis use the OLE Automation programming interface to parse the PowerPoint and Word object model and extract information about document attributes, used fonts, filenames of symbolically linked objects, and application names of physically embedded objects as outlined in Table 1. Document attributes that were extracted include the document's title, creator, creation date, and the application name with which the document was created. For the fonts, only the face name was extracted and for media clips, media type (i.e. whether the clip is a video or an audio file) and the name and path of the media files were extracted.

Analysis of embedded audio and video clips from within PowerPoint or Office document is limited. While the media file format can be reliably identified using, if available, the extension of the media file's name, Codec information is not available within a PowerPoint or Word object model. Thus, the audio and video files need to be analyzed directly to establish any requirements for media players to successfully play back the embedded clips. Media files contain Codec information that media players must determine in order to apply the correct decoding algorithm at the time the media file is played back. However, unless the media file format is known, parsing the media files to extract the Codec information is as difficult as parsing a Word or a PowerPoint document. As a result, an approach similar to Office documents was pursued to parse media files. However, this approach proved unsuccessful because media players provide no or insufficient COM/OLE automation support to load media files into memory and to parse their object model. An alternative approach is to use off-the-shelf freeware tools for media file analysis.

To analyze media files and determine Codec requirements, two products, VideoInspector [3] and AVICodec [1], were used. Although both tools provide Codec information, they differ in the information they provide and in the media files

they are capable of processing. For example, VideoInspector provides information on the number of frames per second (FPS) while AVICodec provides FPS and the total number of frames in the clip. As to the tools capability of processing different media formats, VideoInspector processes QuickTime but AVICodec does not. Conversely, AVICodec processes ASF files but VideoInspector does not. Thus, by using both tools, all downloaded media files from the library were processed successfully. Both, VideoInspector and AVICodec required users to use specific graphical user interfaces to select media files from a target directory and to export the results of the analysis into text files used for subsequent analysis.

Besides software tools for document and media analysis, additional software was built to process artifacts from a library, to search for Office documents and embedded media files on the Web, to build font databases, identify installed fonts on a computing platform, and compare fonts in a font database using font metric information extracted from font files via existing analysis tools. The role and use of this software will be discussed in the experimental study section. The section will start with discussing how Office documents were collected from the Web. It will then present results from the requirement analysis of the document collection and the results from a follow-up experiment whose goal was to study if fonts, missing on a computing platform, can be downloaded from the Web.

4. EXPERIMENTAL STUDY

An experimental study was conducted to determine the effectiveness of the software tools introduced above for establishing software and resource requirements and to identify issues building emulation environments that may be caused by software needs. In the study, approximately 200,000 documents were downloaded from the Web using the Google Web interface to automatically build a collection that contains approximately half Word and half PowerPoint documents. To download the documents, keywords from different glossaries were used to generate queries for searching Office documents on the Web. Each query was composed of a single keyword and a file type specification to match PowerPoint or Word documents only within Google's index. Among the matching results, up to 100 documents were downloaded using the URLs provided by Google. Information about the documents including the URL, a document description as provided by Google, file type information, and the name of the glossary from which the query keyword was selected was stored in a database for subsequent analysis.

To establish visualization requirements for a diverse set of documents, four glossaries consisting of keywords from a science, social science, business/law, and art domain were used. The keywords were collected from free online glossaries published by individuals, educational institutions, or companies. Each document was classified by the name of the glossary from which the keyword was selected in the search. While the approach does not guarantee that a document actually represents information pertaining to the assigned category, we believe it is sufficient to retrieve a diverse set of documents. Table 2 summarizes the information about the glossaries and the distribution of the downloaded Office documents pertaining to the different glossaries. The file sizes of the Office documents differed not only in terms of the file type but also in terms of the glossary. The average file size for PowerPoint documents was approximately 1 MB, ranging from 1.12 MB file size for science and art files to 0.64 MB for social science files and 0.92 MB for business files. For Word documents, the average file size was 0.18 MB. The size difference among PowerPoint documents is due to differences in presentation style. An analysis of the files revealed that science presentations included more pictures and other embedded objects such as graphs, equations, etc. than business or social science presentations. In general, embedding media files affects the file size very little because the files are only symbolically linked and not physically embedded into the Office document.

Prior to processing, all files were scanned for viruses using a standard virus scanning software to protect the platform, artifacts, and the network. Although occurring rarely (less than 30 viruses were detected in 200, 000 Office documents), it takes only a single virus to cause irreparable damage to software and artifacts of a platform.

	glossary size	**#PowerPoint files**	**#Word files**
science	563	39,771	41,596
social science	368	22,582	30,770
business/law	563	28,177	37,808
art	225	6,050	14,195
total	1,719	96,580	124,384

Table 2: Information about the downloaded Office documents and the glossaries.

4.1 Document Analysis Results

The previously discussed software suite was used to analyze the PowerPoint and Word documents in the collection. The analysis was performed on a PC with a Pentium 4, 2.6 GHz processor and 2 GB memory. The processing time for each document was measured using standard OS library calls from within a Python script.

Most files were successfully processed. Only 48 files among the PowerPoint documents and 204 files among the Word documents could not be processed due to errors in the documents or unresolvable problems with the COM/OLE interface. The processing time ranged from 5.6 to 8.7 seconds per file for PowerPoint and 6.16 to 8.24 seconds for Word documents depending on the document's category. The difference in processing time is largely due to the differences in document style. Since most of the time is spent on font analysis, PowerPoint presentations that favor text over embedded objects such as images, video clips, etc. require more time to process than presentations with less text but more embedded objects. Although size is a factor in processing time, differences in the ratio of text versus embedded objects are more significant to explain differences in processing time.

Even if we assume the worst case scenario of 8.7 seconds processing time per PowerPoint presentation and 8.24 seconds for Word documents, the processing time is fairly low making it feasible to process very large document libraries containing millions of Office documents to establish resource and software requirements. In addition, the problem lends itself to conduct the document analysis in parallel on a farm of machines, thereby reducing the total time needed to analyze a large collection of Office documents significantly. Table 3 summarizes the performance statistics with $\oslash T$ and

$\oslash S$ denoting the average processing time and the average file size respectively.

	PowerPoint		Word	
	$\oslash T$ (sec)	$\oslash S$ (MB)	$\oslash T$ (sec)	$\oslash S$ (MB)
science	5.65	1.12	6.40	0.20
social science	7.50	0.64	6.16	0.19
business/law	8.71	0.68	7.08	0.15
arts	8.07	1.06	8.24	0.26

Table 3: Performance analysis of processing PowerPoint and Word files.

Document Attributes: Among the extracted document attributes, both application names and creation date were considered for the collected documents. The results showed that the extracted information was not reliably indicating the documents' creation date or the version of the Office software with which a document was created. While the earliest extracted creation date from Office files was long before the first release of Microsoft Office software, indicating an erroneously stored creation date, version information of the Office software was only specified in some of the extracted application names. Among the application names that indicated version information are "microsoft powerpoint hebrew edition", "microsoft powerpoint 7.0", and "microsoft powerpoint 4.0". However, for the majority of extracted application names, no version information was specified in the application name. While extracting version information may be desirable, it is not necessary to match Office documents with the corresponding Office software since Microsoft Office is backward compatible supporting full access to all Office documents created with older versions of its software.

Fonts: Consolidating the extracted information on fonts showed that both Word and PowerPoint documents use a large variety of fonts with the total number of different face names detected in the entire document collection exceeding more than 3,000. In general, Microsoft supplies 131 different fonts with Office 2003 installed on a Windows XP platform. Thus, the vast majority of detected fonts must come from specific font libraries or software applications other than Office or the operating system. For the remainder of this document, such fonts are considered non-standard Microsoft fonts. Among such fonts, less than 100 were detected to have non-western face names (as determined by the string being in unicode), with only 98 Word documents and 727 PowerPoint presentations using such fonts.

A closer analysis of the font usage revealed that (1) the majority of the fonts (56% of the fonts detected in PowerPoint presentations and 58% of the fonts detected in Word documents) occur only in one presentation or document, and (2) standard Windows platform fonts occur more frequently than non-standard fonts. For example, in Word documents, *Times-New Roman*, the most frequently used standard font for Windows platforms, occurs in 89,629 files, while *Helvetica*, the most frequently used non-standard font (the font is available on Apple Macintosh but not Windows platforms), occurs in 2,583 files. For PowerPoint presentations, the most frequently standard font *Arial* occurs in 54,134 files and the most frequently used non-standard Windows font, *Helvetica* occurs in 3,470 files. This suggests that standard Windows fonts are still favored by authors of Office documents and that non-standard Windows fonts may also be due to Office documents being produced on platforms other than Windows such as Mac OSX.

Media Files: Media files such as audio and video clips occurred predominantly in PowerPoint presentations. Only a single Word file included an image that required QuickTime and a specific encoder to be visible. No media file was physically embedded into the Office documents. The analysis of the PowerPoint presentation returned a total of 6,337 audio and video files that were symbolically linked into the presentations. As previously discussed, the symbolic media link provides the filename with file extension that can be used to successfully classify the media files. A total of 32 different extensions were identified, classifying all of the audio and video clips into 12 different media types. The most common detected media types include QuickTime, MPEG, AVI, Waveform audio, and MIDI; other types include animated images, Macromedia flash, Windows Media/Audio, and audio streams.

The link information itself is insufficient to identify the necessary Codec information for successfully playing back embedded audio or video clips. Thus, in an effort to identify Codec information, the embedded media files were searched on the Web using a URL constructed from the extracted name of the media file and the URL of the PowerPoint presentation that contained the media file. This approach returned a total of 1,117 media files that were subsequently analyzed for Codec information using the previously discussed freeware tools. The results revealed that the media files required a total of 7 different audio Codecs and 25 different video Codecs to be played back with the majority of files requiring MPEG 1, Wave, Radius Cinepak, Microsoft Video 1, and Intel Indeo R3.2 support.

Embedded objects: Among the embedded objects other than video and audio clips, PowerPoint presentations contained 435 and Word documents 174 embedded objects such as Excel spreadsheets or graphs, Visio graphs, images, Microsoft WordArt, diagrams and more. When considering documents from the four different categories as well as the number of files in each category, we found that most embedded objects occurred in art documents while business/law documents contained the fewest number of embedded objects. All of the embedded objects were visible when inspect in the presentation or document but, as expected, could not be edited absent of the application that generated them. The most frequently used embedded objects were Microsoft WordArt, JPEG and GIF images, drawings, diagrams, and spreadsheets. Only a single Office file contained a Visual Basic Macro which performed a database call to retrieve data to be shown in the document.

4.2 Building Font Databases

The large number of missing fonts prompted us to conduct an experiment to investigate whether it is possible to automatically build font databases by searching for fonts on the Web using a font's face name as a query term. Unfortunately, this strategy proved to be unsuccessful for several reasons. First and foremost, fonts that are missing may not be published on the Web. Second, as previously discussed, a font's face name alone is insufficient to identify a font. Thus, a missing font may be published on the Web but due to naming differences cannot be located. Nevertheless,

among the 1,233 downloaded fonts, 27% matched missing fonts in PowerPoint presentations and 18% matched missing fonts in Word documents judged by comparing the fonts' face names.

To search for fonts on the Web, first a list of face names was compiled from the list of missing fonts that were frequently used in both Word and PowerPoint files. Using each face name and a file type specification to match font files, a Google query was generated to search and download fonts on the Web. Among the matching results, up to 10 true-type fonts were downloaded using the URL provided by Google. The corresponding files were stored in a font database along with information about font attributes that was automatically extracted from the font files. For the automatic font analysis of downloaded true-type fonts, a publicly available software from Microsoft was used [5] to extract font attributes from the header of the font file. Some of the font attributes that were considered include the average char width in x dimension, the panose (a 10 byte series of numbers specifying the visual characteristics of a given typeface), the typographic ascender and descender of the font, and more. In total, 20 font attributes were extracted from a font's header file that were sufficient to distinguish and compare the downloaded fonts (a full list of available attributes is available at [4]). Approximately 1,300 fonts were searched and 1,233 different fonts were collected and stored in the database. The downloaded fonts were subsequently compared against the list of missing fonts in PowerPoint and Word files using the font face name. The comparison revealed that 334 fonts matched missing fonts occurring in PowerPoint and 255 fonts matched missing fonts occurring in Word files. When performing an approximate match in which the face name of the downloaded font partially matches the face name of missing fonts, an additional 475 fonts matched fonts occurring in PowerPoint and an additional 481 fonts matched fonts occurring in Word files.

In a separate experiment, frequently missing fonts used in both Word and PowerPoint documents where searched in a font database provided by Adobe [2]. This database contains approximately 4,200 fonts and is used by Adobe to convert Office documents into PDF files for its online PDF conversion service. When searching the font database, a total of 333 out of 1,994 missing fonts where found. This number increases to 1,003 fonts, when performing an approximate match in which the face names of the missing fonts partially match fonts in the Adobe font database. The results show that even large font databases used by an industry that specializes in content creation and delivery cannot guarantee availability of frequently used fonts to correctly visualize collections of Office documents.

Fonts cannot be distinguished on the basis of the font's face name only. Preliminary results have shown that many fonts have different face names but are visually identical. To determine the similarity or difference of the downloaded fonts to each other, a simple clustering algorithm was applied that uses the previously mentioned font attributes to group the fonts. The algorithm assigns fonts to the same group if their font attributes have identical values. The analysis showed that among the 1,233 font files, 1,041 fonts were different. Examples of were fonts were identical but had different face names include *Mead Bold* and *Andy* or *Arial* and *TITOL Arial Narrow*. However, for the majority of grouped fonts, the fonts of a group shared the same face names.

Rather than searching for fonts on the Web, a more direct approach is to require font files to be submitted along with the artifacts. However, this will require artifacts to be submitted to long-term repositories from the same platform where they are originally authored. Once, the artifact is moved from its authoring platform, there is no guarantee that the new platform supports the same set of fonts as used by the artifact. Another alternative to address the font issue is to require fonts to be embedded into the artifact. This, however, would increase the size of the artifact substantially and unnecessarily, especially if the embedded fonts exist on the platform on which the document is to be stored.

5. AUTOMATIC TEST CASE CONSTRUCTION

An emulation platform must be tested to ensure that future attempts to access and interact with a digital artifact does not produce errors or anomalies due to inadequacies in the emulation platform. For example, when viewing a PowerPoint presentation, the application should not crash, and all visual and audio effects should be rendered with reasonable fidelity. Of course, some artifacts in a collection may be inherently flawed. Testing emulation platforms poses several technical challenges. First, a set of tests must be defined that, when applied to a single artifact on an emulation platform, ensure the correct behavior of that artifact. Second, a test strategy must be developed which ensures that if each test is applied to a selected subset of the artifacts, the correct behavior is guaranteed for all artifacts with high probability. Third, the tests and associated strategy must be implemented with reasonable computational and human resources. In this paper, we are focusing on the first and third challenge and leave the second for future research.

Representative test cases for manual inspection must be created to cover the different resource needs of the digital artifacts in the library. The test cases must be few in number and small in size to make human inspection feasible. An inspector of a test case must check the digital artifacts for their correct visualization and functionality. For example, the test cases for PowerPoint must include slides that contain video and audio clips corresponding to the different video and audio formats and Codecs used by the PowerPoint presentations in the digital library. To construct representative test cases, the results from the requirement analysis may be used to compile a list of fonts, video and audio clips, and embedded objects to be examined to guarantee proper visualization and functionality of the artifacts in the digital library.

To support testing of an emulation environment, we investigated automation of test case construction. As a first step toward interactive testing of emulation environments, prototype software tools were developed that use output files from a requirement analysis to automatically build PowerPoint presentations for testing proper playback of media clips. To simplify manual inspection, each slide contains a single audio or video clip of a different format and with a different Codec. The slides' titles display the name of the media file and the Codec to allow inspectors to quickly identify if the media format and Codec is supported within PowerPoint. When the software tools were applied to generate a test case to determine support for various types of audio and video clips on a Windows XP computing plat-

form, a presentation with 34 slides was constructed. The presentation covers the different audio and video clips and Codecs that occur in the PowerPoint presentations of the document library as discussed above.

The (generated) PowerPoint test presentation was easy to inspect in viewing mode. Any problems with playing back embedded media files were easily and quickly identified. For example, it was quickly determined that any QuickTime movie that appeared in the presentation could not be played back successfully under Windows XP even though QuickTime was installed and the same movie played back correctly outside PowerPoint using the QuickTime movie player directly. In contrast, when inspecting the same presentation under Mac OS X, playing back any embedded QuickTime movies from within PowerPoint was not an issue. In total, 10 of the 34 embedded movie and audio clips did not play back correctly within PowerPoint due to missing Codecs or Windows specific issues related to the QuickTime plug-in for PowerPoint.

6. DISCUSSION

A few problems were encountered during the coding phase and the experimental study of this research project. They include (1) issues related to the large variety of fonts discovered in Office documents, (2) deficiencies of the object model to extract information about resource requirements, and (3) performance issues when using the OLE/COM interface to parse the object model.

If documents are accessed on platforms that don't supply the necessary fonts, Microsoft Office substitutes the missing fonts with existing fonts causing, at the minimum, formatting problems with the document. However, in some cases, font substitution may result in a loss of information when the font itself carries specific meaning as is the case for symbol fonts such as *Symbol*, *Bookshelf Symbol 7*, or *MS Reference Specialty*. Users are typically not informed when fonts are automatically substituted with available fonts causing confusion when formatting issues arise in an opened document.

The large variety of fonts used in documents poses a challenge to digital preservation. Many fonts that were detected cannot be distinguished by the face name itself because fonts may have different face names but may be identical in their visual characteristics. This is an issue that largely remains unresolved because it requires outside intervention to be addressed. For example, Office documents must either embed non-standard fonts in the files or encourage users to use standard fonts when the artifacts need to be stored in document libraries for long-term preservation.

Our approach to use the OLE/COM interface to perform a requirement analysis of Office documents has been mostly successful. Performance issues occurred when Word documents were scanned character by character to extract font information. Access and processing problems occurred because of file access restriction, file corruption, or embedded macros. In the latter cases, some form of user intervention was needed to continue with an automatic document analysis. For example, when access is restricted, a dialog box opens up, requiring users to enter a password. To address performance problems with Word documents, paragraphs instead of characters were scanned to extract font information from the file. Preliminary experiments have shown that scanning paragraphs is sufficient to determine the fonts used in a document. Access and processing problems were resolved by terminating and logging any files that could not be analyzed within a fixed time period. In general, the number of processing problems were small enough for a manual examination to be feasible given the large number of documents that were processed.

A critical component in the proposed architecture is the operating system simulator. We believe that the technology for creating and preserving machine simulators is well understood and should be not considered a weak spot in the proposed solution. Indeed, machine simulation is widely used for system design [19, 11], software testing, and cross platform software support [22, 9, 10]. Much of the writing on emulation as a digital preservation strategy has focused upon the fear that we may lose the ability to execute the machine simulator unless it is captured in some formal language [14]. We believe that this is not the most serious problem relating to emulation; indeed small teams have demonstrated the ability to build hardware emulation platforms to support complex systems in relatively short periods [10]. Emulation also presents a serious usability issue; it seems likely that some additional helper applications should be included in an emulation environment to enable future users to successfully launch and utilize an artifact in an emulation environment without extensive experience with the underlying software. Although this is an important issue, it is out of the scope of this paper.

Using four different glossaries to download Office documents from the Web proved to be a successful approach in building a document library with a diverse collection of Office files. When comparing files from each glossary category, while taking the number of files in each category into consideration, we found that science and art documents contained more multimedia clips than social science or business/law documents. The fewest multimedia clips occurred in business/law documents. As to embedded objects other than multimedia clips, art documents contained more embedded objects than any other files while business/law, social science, and science documents contained approximately the same number of embedded objects such as images, Microsoft WordArt, graphs and spreadsheets.

Emulation is only one piece of a larger digital preservation problem. An important issue that was not addressed in this paper relates to the organization of digital archives (e.g. such as handled by dspace [21]) as well as storing and preserving the digital "bits." Another issue that was not touched upon is copyright and licensing. For example, many fonts require licensing before they may be used on a client workstation to display the content of Office documents that uses them.

7. CONCLUSIONS

We have proposed to use emulation as a strategy to preserve digital artifacts and facilitate future access. In the context of digital preservation, emulation is generally considered to consist of the artifact to be preserved (the document), the contemporaneous software required to access and interact with the artifact (the software image), and the hardware simulation software required to execute the software image (the emulator). Emulation appears to provide the only technique that can preserve the behavior of arbitrary interactive digital artifacts in the face of hardware obsolescence.

In this paper, we have discussed strategies and presented

a suite of software tools for building and testing emulation environments to support digital preservation. The strategies and software tools have been used in an experiment to analyze a collection of 200,000 Office documents, half PowerPoint presentations and half Word documents. The results showed that authors employ a large variety of fonts in their Office documents that far exceed the standard set of fonts supplied by Microsoft's operating systems and Office applications. The widespread use of non-standard fonts in Office documents is likely to cause formating and visualization problems when accessed on platforms where fonts are missing. The problem can only be resolved if non-standard fonts will automatically be embedded in the documents when stored for long-term usage. On a positive note, analyzing the requirements for embedded media files and other embedded objects was straightforward and scaled well with large document libraries. Any embedded objects other than media clips and audio/video streams that occurred in the document library were correctly visualized upon inspection. For audio/video clips, a relatively small number of Codecs and media types were identified making it possible to collect the necessary software and resources to build emulation environments that support correct visualization of the embedded media clips. Also, initial prototype software for automatic construction of test cases to examine an emulation environment's robustness has been successfully applied and proven to be useful.

8. ACKNOWLEDGMENTS

We wish to express our thanks to Google for allowing us to access the Google index to search for and download Office documents and font files.

9. REFERENCES

[1] AVICodec. http://avicodec.duby.info/.

[2] Fonts supported by Create Adobe PDF Online. http://www.adobe.com/support/techdocs/328731.html.

[3] KC Softwares. VideoInspector. http://www.kcsoftwares.com/.

[4] Microsoft OpenType fonts specification. http://www.microsoft.com/OpenType/OTSpec/os2.htm.

[5] Microsoft Typography: Internal Development Tools. http://www.microsoft.com/typography/tools/tools.aspx.

[6] Microsoft PowerPoint Object Model. In *MSDN Library*, 2001.

[7] F. Bellard. QEMU. http://en.wikipedia.org/wiki/QEMU.

[8] K. Brockschmidt. *Inside OLE (2nd ed.)*. Microsoft Press, Redmond, WA, USA, 1995.

[9] J. C. Dehnert, B. K. Grant, J. P. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson. The transmeta code morphing software: using speculation, recovery, and adaptive retranslation to address real-life challenges. In *Proceedings of the international symposium on code generation and optimization*, pages 15–24. IEEE Computer Society Press, 2003.

[10] G. Desoli, N. Mateev, E. Duesterwald, P. Faraboschi, and J. A. Fisher. Deli: a new run-time control point. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 257–268. IEEE Computer Society Press, 2002.

[11] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homewood. Lx: a technology platform for customizable vliw embedded processing. In *Proceedings of the 27th annual international symposium on Computer architecture*, pages 203–213. ACM Press, 2000.

[12] S. Gilheany. Preserving digital information forever and a call for emulators. In *Digital Libraries Asia 98: The Digital Era: Implications, Challenges, and Issues*, 1998.

[13] A. R. Heminger and S. Robertson. The digital rosetta stone: a model for maintaining longterm access to static digital documents. *Communications of the AIS*, 3(2), 2000.

[14] R. A. Lorie. A methodology and system for preserving digital data. In *Proceedings of the second ACM/IEEE-CS joint conference on Digital libraries*, pages 312–319. ACM Press, 2002.

[15] A. T. McCray and M. E. Gallagher. Principles for digital library development. *Communications of ACM*, 44(5):48–54, 2001.

[16] P. Mellor. CaMiLEON: emulation and BBC doomsday. *RLG DigiNews*, 7(2), 2003.

[17] S. Miastkowski. Create two virtual PCs Out of One. *PC World*, 1999. http://www.vmware.com/news/articles/1999.html.

[18] Microsoft. Virtual PC. http://en.wikipedia.org/wiki/Virtual_PC.

[19] F. A. Salomon and D. A. Tafuri. Emulation - a useful tool in the development of computer systems. In *Proceedings of the fifteenth annual simulation symposium*, pages 55–71, 1982.

[20] R. Stallman. GNU Debugger. *GNU General Public License*, 1999. http://en.wikipedia.org/wiki/GDB.

[21] R. Tansley, M. K. Smith, and J. Harford Walker. The DSpace open source digital asset management system: Challenges and opportunities. In *Lecture Notes in Computer Science 3652: Research and Advanced Technology for Digital Libraries: 9th European Conference*, pages 242–253, Vienna, Austria, September 2005.

[22] S. G. Tucker. Emulation of large systems. *Communication of the ACM*, 8:753–761, 1965.