Using Emulation to Preserve Digital Documents

Jeff Rothenberg

RAND-Europe

Koninklijke Bibliotheek
The Hague, July 2000

Cover design
Geert Henderickx

Illustration
Frank Dam

Available from
Koninklijke Bibliotheek
PO Box 90407
2509 LK The Hague
E-mail info@kb.nl

ISBN
906259145-0

© RAND-Europe / Koninklijke Bibliotheek, The Hague

This study was commissioned by the Koninklijke Bibliotheek, national library of the Netherlands, and jointly funded by the Dutch programme 'Innovation in the provision of scientific Information' (IWI)

TABLE OF CONTENTS

PREFACE / 4

SUMMARY / 5

1. INTRODUCTION / 8

2. WHY IS IT HARD TO PRESERVE DIGITAL DOCUMENTS? / 9
   - 2.1 Logical Format / 9
   - 2.2 Software-dependence / 12
   - 2.3 Dynamic, interactive, and executable documents / 13
   - 2.4 Authenticity / 15
   - 2.5 Cannot 'just save' digital documents / 16
   - 2.6 The problem of preserving digital documents / 17

3. IMPLICATIONS OF DIGITAL DOCUMENTS AS PROGRAMS / 20

4. THE CONCEPT OF EMULATION / 24
   - 4.1 Emulating hardware is easier than emulating software / 25
   - 4.2 Relevant traditional uses of emulation / 27

5. USING EMULATION TO PRESERVE DIGITAL DOCUMENTS / 30
   - 5.1 What would emulation achieve? / 30
   - 5.2 Emulation versus migration / 32

6. HOW WOULD EMULATION WORK? / 37
   - 6.1 Creating emulators for future computers / 39
   - 6.2 Hardware configurations, versions, and modular emulation / 43

7. UNRESOLVED QUESTIONS AND ISSUES / 45

8. CONCEPT OF OPERATIONS FOR EMULATION-BASED PRESERVATION / 47
   - 8.1 Initial production of digital documents to be preserved / 47
   - 8.2 Acquisition (ingestion) of documents for preservation / 49
   - 8.3 Ongoing preservation activities / 50
   - 8.4 Providing access to preserved digital documents / 52

9. COST COMPARISON / 54

10. REQUIRED RESEARCH / 57
    - 10.1 Emulator specification formalism / 57
    - 10.2 Human-readable annotations and explanations / 58
    - 10.3 Encapsulation techniques / 59
    - 10.4 Develop 'helper' programs to aid in using old digital documents / 61

11. CONCLUSIONS AND RECOMMENDATIONS / 62

BIBLIOGRAPHY / 64

PREFACE

This report considers the problem of how to preserve digital documents, given the fact that their formats quickly become obsolete, as do the programs that originally interpreted those formats and the computers on which those programs ran. This problem is investigated from the perspective of the deposit library community, though the issues and solutions discussed here also apply more broadly to the full range of digital data, documents, records, and other artifacts that are used by other kinds of libraries, archives, government agencies, commercial organizations, and individuals.

The report attempts to identify and illuminate the root of this problem and, more specifically, discusses the theoretical and practical issues involved in using emulation (a proven computer science technique in which one computer is used to reproduce the behavior of another computer) as a way of preserving authentic, accessible, usable digital documents. Although it is aimed at the deposit library community, this discussion should also be of interest to members of the general library community, as well as archivists, government recordkeepers, and preservationists concerned with the great potential for loss of information that has emerged as an unexpected and unfortunate aspect of the digital age.

SUMMARY

The increasing use of digital technology to produce documents, databases, and publications of all kinds has led to an impending crisis resulting from the absence of available techniques for ensuring that digital information will remain accessible, readable, and usable in the future. Deposit libraries – as well as other libraries, archives, government agencies, and organizations – must find ways to ensure the longevity of digital artifacts or risk the loss of vast amounts of information and human heritage.

While it is now generally understood that digital information must be copied to new storage media quite frequently, since such media become obsolete in a few years, there is a deeper problem that must be solved as well. Digital information can be rendered usable only by running appropriate software, and such software – along with the hardware on which it runs – can become obsolete just as quickly as the media on which digital information is stored. The software that must be run to make a given digital document usable must understand the 'logical format' of the document, which is what enables the document to be made intelligible. Without appropriate software, trying to read a digital document is like trying to read hieroglyphics without a Rosetta Stone.

In fact, a digital document can be thought of as a program, which must generally be interpreted by some other application program (running on some computer) in order to be meaningful to a human reader. Since such application programs often become obsolete in just a few years, the key question for preservation is how to read a digital document once the original application program that interpreted it (or the computer on which that program runs) has become obsolete.

The solution most often proposed to this problem involves 'migration' (i.e., conversion) of a document from its original logical format into successive subsequent formats as each format becomes obsolete. This is highly labor-intensive, since if it is not to be lost, *every* document must be converted in this way every time its current logical format becomes obsolete, whether or not it is likely to be accessed prior to the next required conversion. More fundamentally, each conversion entails a significant risk of corrupting a document, possibly sacrificing its original appearance, structure, interactive

behavior or 'look-and-feel' if not its actual content. Further, since each conversion is performed on the result of the previous conversion, this corruption is cumulative, nor can the original be used to correct or even detect such corruption, since the original is by hypothesis no longer usable shortly after the first conversion occurs.

Whether this corruption has a meaningful effect on preserved documents depends on what aspects of such documents are considered essential to their authenticity. Authenticity criteria may vary both across disciplines and potentially across different types or uses of documents. By analogy with traditional documents, a 'digital-original' can be defined as any representation of a digital document that has the maximum likelihood of retaining all meaningful and relevant aspects of the document. The most obvious candidate for a digital-original of a document is the original logical format of that document, preserved in such a way that it can be perpetually interpreted by its original, intended interpreter.

This amounts to requiring that the original application software for a given document's logical format remain executable indefinitely. Emulation (in particular, using software to allow future computers to reproduce the behavior of obsolete computers) appears to be the most promising of the possible ways of making such original software executable far into the future. Emulation is a widely used technique, in which one computer system reproduces the behavior of another system. Though it has not yet been applied to preserving digital documents in any systematic way, neither has any other approach. Emulation is the only approach yet proposed that promises to preserve digital-original documents and is based on proven technology. In addition, emulation can be tested in advance on individual documents to demonstrate (with reasonable certainty) that they will be properly preserved.

Accessing a digital-original document preserved via emulation would require readers to use its original, obsolete software in an (emulated) obsolete hardware environment. This would be analogous to reading an ancient manuscript in its original form, which requires scholarly expertise that few readers possess. However, the emulation approach could make preserved digital documents available in future 'vernacular' versions by extracting such versions from digital-originals at any time. While this requires a conversion similar to migration, this process would (i) not accumulate corruption errors, since conversion would be performed directly on the original each time, (ii)

not involve the urgency of migration, which must be performed before the original becomes obsolete, (iii) allow direct comparison of each vernacular version against the original (which migration cannot allow, since the original becomes unusable after its first conversion), and (iv) maintain a digital-original for scholarly, legal, and historical use (which migration cannot offer).
Emulation-based preservation could be implemented immediately by writing individual programs to emulate each obsolescent computing platform on its successor. This emulator would allow rendering on the new platform all documents that used any software that ran on the older platform once it becomes obsolete. As the successor platform itself becomes obsolete, a new program would be written to emulate it on its own successor in turn, and older emulators would be run under newer ones. This 'layered emulation' approach would require no new technology but would be somewhat inefficient. Alternative emulation approaches (requiring some additional research) could use a formal emulation specification language to describe older platforms. Interpreters of this language running on future platforms could then emulate any previous platform without requiring individual programs to emulate each older platform on each future platform and avoiding the recursive aspect of layered emulation.
Finally, an 'emulation virtual machine' approach would allow a specification language interpreter to be ported to new platforms with a minimum of effort. Using this approach, documents to be preserved using emulation would be tested for completeness, after which appropriate metadata would be generated, and software, hardware specifications, and metadata would be encapsulated with the document bit streams, to be copied to new storage media as required. Beyond this effort (much of which would be required by any preservation approach) emulation would require merely porting the emulation virtual machine to each new computing platform and writing an emulation specification of that platform before it becomes obsolete. The cost of this approach to digital preservation should be far lower than that of migration, since emulation requires no repeated processing of individual documents.

1. INTRODUCTION

The rapid adoption of digital technology for producing documents, databases, and publications of all kinds has been driven by the unprecedented capabilities for creating, accessing, and processing digital information that are afforded by this new technology. Yet the transition to digital publication has led to an impending crisis resulting from the absence of any available techniques for ensuring that digital information will remain accessible, readable and usable in the future.[1] Digital artifacts from the past few decades are already often difficult or impossible to use, since the programs required to access or interpret them – and the computers that are required to run those programs – are already obsolete. In this way, the extraordinary pace of progress in information technology undermines the longevity of the very digital artifacts that it encourages us to create. The advantages of the digital transition are too compelling to be sacrificed, but the ultimate cost of this transition may be the loss of vast amounts of information and human heritage unless we can find a way to ensure the longevity of digital artifacts. This report considers this problem from the perspective of the deposit library community, in which digital 'publications' such CD-ROMs or online 'e-journals' are currently the greatest concern. The criteria that deposit libraries use for evaluating whether publications are authentically preserved are derived from their interactions with publishers and may be different from the equivalent criteria adopted by archivists or even other kinds of libraries. For the most part, however, the issues and solutions discussed here also apply more broadly to the full range of digital data, documents, records, and other artifacts that are used by other kinds of libraries, archives, government agencies, commercial organizations, and individuals. Some issues (such as the criteria for the authenticity of preserved digital artifacts or the timescales over which they must be preserved) may depend on the type of artifact in question, the organization or person using it, or the use to which it is to be put; however, in order to address the problem in its most general form, this

[1] With a few exceptions, specific citations of supporting literature and related work will not be provided in this report, in keeping with the narrative nature of the discussion. However, a bibliography of relevant items is provided at the end of the report for readers who would like to pursue the subject further.

report avoids such distinctions except where they are relevant.[2] The term 'digital document' will accordingly be used throughout to denote the general case, while other types of artifacts (e.g., publications or data) will be distinguished only where necessary. [3]

The intent of this report is to identify and illuminate the root of the digital preservation problem and, more specifically, to discuss the theoretical and practical issues involved in using emulation (in this case, enabling future computers to reproduce the behavior of older computers) as a way of preserving authentic, accessible, usable digital documents, while explaining what would be entailed in implementing a digital preservation strategy based on this approach.[4] Note that although emulation is a well-proven technique, since it has not yet been applied to the preservation of digital documents in any systematic way, much of this report is necessarily hypothetical.[5]

The next section discusses the nature of digital documents and explains why preserving them poses such a problem; it argues that digital documents are actually programs that must be interpreted by software to be made usable by humans. Section 3 explains the crucial computer science notion of 'interpretation' in further detail, focusing on the distinction between hardware and software interpreters. Section 4 explains the related concepts of emulation and virtual machines, as well as describing traditional uses of emulation that are relevant to preservation. Section 5 then suggests how emulation could be used to preserve digital documents. Section 6 presents several alternative technical approaches to implementing such a preservation scheme, while Section 7 discusses remaining issues and questions. A 'Concept

---

[2] A digital artifact can actually be thought of has having several different types of 'type' based on its form (e.g., memo, paper, report, book), its content (e.g., text, graphics, multimedia), its function (e.g., message, publication, record), or its context of use (e.g., legal, artistic, informational).

[3] Although the term 'artifact' is perhaps more generic than 'document' the latter seems less awkward.

[4] The computer science term 'emulation' denotes a process in which one computer system is used to reproduce the behavior of another system with such fidelity that the emulation can be used in place of the original system, at least for most purposes.

[5] It is important to note, however, that no other technique has yet been applied to the preservation of digital documents in any systematic way either, including 'migration' (which, as discussed below, is the most frequently proposed alternative to emulation).

of Operations' is offered in Section 8, showing how a particular emulation approach might be used by a library or other preservation agency to preserve and access digital documents over time. Section 9 discusses the expected costs of emulation-based preservation and argues that these would be considerably lower than the costs of migration. Finally, Section 10 outlines additional research that should be performed to develop an ideal emulation approach, and Section 11 presents conclusions and recommendations.
Before discussing emulation, we begin by explaining the need for a new approach to preserving digital documents.

2. WHY IS IT HARD TO PRESERVE DIGITAL DOCUMENTS?

2.1 Logical Format

It is widely recognized that a digital document will become inaccessible if the medium on which it is stored becomes obsolete and unusable before the bit stream representing the document is copied to some other medium. It should be just as obvious that a digital document can become unintelligible if the 'logical format' of its bit stream (i.e., the structure and encoding of the sequence of bits in that stream) becomes obsolete and unusable before the document is converted into some other format. In fact, this latter kind of obsolescence is more difficult to counteract than the former. A given bit stream can be copied without loss from one digital storage medium to any other, but one logical format cannot in general be converted into a different logical format without a significant chance of loss or corruption. Whereas identical bit streams represented on different storage media represent identical information, no such equivalence can be assumed between different logical formats.

This implies that the solution to the storage medium obsolescence problem (copying to a new medium before the old one becomes obsolete) is a naive model for solving the problem of logical format obsolescence. Merely 'copying' (i.e., converting) a digital document from one logical format to another as the old one becomes obsolete entails a significant risk of corrupting the document. Yet this is the basis of the 'migration' approach to digital preservation. Migration is a traditional computer science technique that has been used for decades to keep data, documents, records, and programs accessible and usable. However, it is widely regarded by the data administrators, data processing personnel, and computer scientists who have employed it as a labor-intensive, time-consuming, expensive, and error-prone process.

The 'logical format' of a bit stream implicitly specifies its intended interpretation. For example, the same stream of bits may be interpreted as a string of textual characters, a photographic image, a musical phrase, an animated graphical sequence, etc. It is in general impossible to tell from a bit stream how it is intended to be interpreted, just as it was impossible to decipher certain streams of hieroglyphic symbols prior to the discovery of the

Rosetta Stone. It is no more possible to make a bit stream self-explanatory than it would have been to make hieroglyphics self-explanatory by providing explanations written in hieroglyphics.[6]

2.2 Software-dependence

More fundamentally, digital documents are designed to be interpreted by computer programs prior to being viewed by human readers. Not only are such programs required to extract the intended sequence of bits from a physical bit stream, but they are essential for interpreting the resulting logical bit stream as well.[7] Digital documents are by their nature software-dependent. While it is tempting to imagine that we might characterize and standardize logical formats so well as to make them intelligible without the use of specific programs, this strategy has many limitations. For one thing, the set of important and popular logical formats continues to evolve rapidly, and this evolution shows no sign of abating.[8]

---

[6] In fact, bit streams are harder to interpret than text, since there are no spaces, punctuation, or other separators between groups of bits to indicate where individual characters, words, or other 'chunks' of information begin and end. And although text can generally be assumed to be a linear sequence of characters or words, many logical formats utilize 'pointers' embedded in their bit streams to link non-adjacent groups of bits into logical sequences that differ from their physical sequences. Yet pointers look just like any other strings of bits; without prior understanding of the pointer conventions used in a given logical format, it is impossible even to determine the logical sequence of bits (i.e., the 'logical bit stream') that a physical bit stream is intended to represent.

[7] Even a simple textual encoding relies on software to map a given bit sequence into a text character. More complex encodings (such as word processing, page layout, graphics, imagery, sound, hypertext, hypermedia, etc.) rely all the more essentially on software to make digital documents intelligible.

[8] Although this chaos of formats sometimes appears to be converging toward a small set of standard forms, such apparent convergence has so far always been short-lived. For example, the emergence of the World Wide Web initially sparked predictions that documents would converge toward the simple text form embodied in HTML, but embedded images, scripts, applets, and the increasing use of 'plug-ins' for specific software-dependent formats quickly reversed that trend. Though XML is undoubtedly an advance over HTML, it seems unlikely that it will be able to stem the natural tide of divergence either.

With a few notable exceptions, each logical format, whatever its intended output, is inextricably bound to a given program that knows how to interpret it. To the extent that the market encourages this tendency by rewarding incompatible, proprietary solutions with increased market-share, it seems unlikely that this trend will reverse in the foreseeable future.[9] It is sometimes possible to extract some of the content of a digital document by the use of something other than its intended software,[10] but this typically causes the loss of certain aspects of the original document, which may compromise its integrity and authenticity. Vendors are motivated to ensure such losses so that customers will be forced to buy their products rather than those of their competitors; but even without this factor, such losses are inevitable. Any extraction or interpretation of a digital document by a program other than the one intended to be used with it constitutes conversion of the document, which entails potential loss and corruption of its content, structure, and meaning.

2.3 Dynamic, interactive, and executable documents

Because the use of computers facilitates the creation and use of evolving documents and other digital artifacts, many of these have become inherently dynamic. That is, agendas, calendars, working drafts, laboratory notebooks, mailing lists, most databases, etc. are by their nature dynamic in that their content is continually changing to reflect the most up-to-date information. In addition, newer kinds of digital documents, such as web pages, are increasingly being designed to tailor themselves to the reader's stated or implied needs or desires, based on previously-established 'user profiles' or

---

[9] The market also encourages compatibility (e.g., in order to allow programs to be used together), but the tendency toward incompatibility seems to be the stronger of the two forces.

[10] The term 'intended software' is used here to mean the software that should be used to correctly render a digital document. While documents do not themselves have intentions - and the intentions of their authors or publishers may be unknown - it is nevertheless clear that most digital documents are intended to be used with one or a small selection of specific programs, as indicated by the fact that they are described by terms that include the names of this software, e.g., Word documents, PowerPoint presentations, JPEG images, etc. The 'intended software' of a digital document is implicit in its logical format.

identifiable attributes, such as the type of system or browser a user is running or the Internet 'domain' through which a connection is made. Digital documents may therefore be dynamic both over time and in response to who is reading them. A dynamic document can be thought of as a generator for a number of alternative documents, that can evolve over time or can be customized to the needs of particular readers.

Moreover, digital documents may be dynamic in the sense of being interactive. Most web pages, CD-ROM publications, and other 'online' documents allow their readers to interact with them to tailor their form, content, or appearance to the needs and desires of their readers as they read them. Interactive documents are more than simply dynamic: they are responsive, changing at the request of their reader.

Finally, all digital documents are essentially programs, which must be interpreted by other programs. That is, the digital artifact that is stored (i.e., the bit stream) is in part or entirely an executable entity. This is most obvious for such digital artifacts as spreadsheets or word processing documents that contain executable 'macros' or for hypermedia documents containing embedded 'scripts' or 'applets' that are executed as part of the process of displaying (or 'rendering') them; however, it is actually more useful to think of all digital documents as programs, as argued below.

The executable, dynamic, interactive and responsive nature of digital documents therefore makes them fundamentally different from traditional documents.[11] The complex behavior of a digital document is contained within it as a potential, only a small part of which is manifested in any given 'rendering' of the document. It is for this reason that saving digital documents in 'rendered' form (for example, printing them) is inadequate as a way of preserving them. Attempting to render such documents using software other than that which was intended – or converting them into different logical formats – is unlikely to result in proper interpretation of their executable content, since the proper behavior of any program is fundamentally dependent on its intended interpreter.

2.4 Authenticity

Whenever a digital document is converted from one logical format into another – or is for any reason interpreted by anything other than its originally intended interpreter – there is a chance that it will be corrupted. However, this corruption may or may not be significant, depending on which aspects of the document's intended interpretation are relevant to retaining its authenticity. For example, if the only meaningful aspect of a text document is considered to be the sequence of words contained in its body, then a conversion that loses or corrupts the document's original layout, structure, pagination, fonts, spelling, punctuation, footnotes, cross-references, citations, etc. may be acceptable. In order to decide what kinds of conversion may be acceptable, it is therefore necessary to define what is meant by preserving a document in its 'authentic' form. This effort would lead to the development of 'authenticity criteria' for digital documents.[12]

Such criteria might be different for different classes of documents or different kinds of use of such documents. For example, casual readers might define less stringent authenticity criteria than scholars or lawyers using documents to assess accountability. In some cases it might be sufficient (or even preferable) to extract modern, converted renditions of a document, whereas in other cases all attributes of the original might be of equal importance, including, for example, the 'look-and-feel' of the document as its original author or readers experienced it.[13] Documents composed of text, graphics, sound, photographic imagery, video, etc. may embody different authenticity criteria depending on the unique attributes of these different types of content, whereas executable,

---

[11] This is especially true for documents that are 'born digital' rather than being digitized from more traditional forms. However, even a digitized document may take on some of these attributes as its digitized form takes on a life of its own.

[12] Note: 'authenticity' is used here in its broadest sense, meaning essentially the validity and suitability of a document for some purpose. Authentication (i.e., the verification that a document is what it purports to be) is only one aspect of authenticity, and while it is of interest in its own right, it is not discussed in this report.

[13] For example, a reader interested in the gender roles implied by a medieval manuscript might prefer a modern transcription of the text, whereas a student of calligraphy might require a reproduction of the original, which preserves its typography.

dynamic, or interactive documents may by their nature demand more comprehensive authenticity criteria than simpler, more traditional documents.

Since at least some libraries of deposit consider it their mission to retain deposited documents without republishing, reformatting, or changing them in any way, their criterion for the authentic preservation of such documents may be that (to the extent possible) future users should be able to see and use those documents exactly as their original audiences did. On the other hand, a data warehouse might require that authentic preservation allow future users to explore implicit relationships in data, even if original users were unable to see (or define) those relationships. Scholars interested in the choices and constraints that older technology presented to authors may choose a somewhat different authenticity criterion, requiring that preserved digital documents retain all of the behaviors that they originally exhibited to their authors. From a different perspective, many archivists argue that archival theory implies that if preserved records are to be considered authentic, future users must be able to understand the roles that those records played in the business processes of the organizations that generated and used them, as well as being able to use those records in any future business processes that may require them (e.g., for determining past accountability).

2.5 Cannot 'just save' digital documents

The 'original' of a traditional, non-digital document is inseparable from the physical medium on which it is written or printed: the medium 'carries' the document. Saving an original document of this sort is therefore equivalent to saving its physical carrier. Furthermore, saving the physical carrier automatically saves all of those attributes of the original that it is possible to save; some of these may fade or otherwise decay over time, but nevertheless, saving the physical original of such a document saves all of its attributes that can be saved.[14]

Yet the situation is quite different for digital documents. In the first place, a digital document is not in any meaningful sense inseparable from the physical

[14] It can be argued that the importance of originals in the traditional sphere stems from the fact that an original document retains all of its attributes that can be retained, thereby satisfying the greatest possible range of authenticity criteria.

storage medium on which its bit stream happens to be stored: in fact a given digital document may be stored simultaneously and interchangeably on many different storage media at once or during its creation or lifetime. The physical storage medium on which a digital document happens to be stored at a given moment is fundamentally irrelevant to the document.[15]

2.6 The problem of preserving digital documents

The independence of digital documents from their storage media is fortunate, since these media become obsolete so quickly. Because they are independent of their carriers, digital documents can (and must) be copied to new media to prevent their loss. But there is another important difference between digital and traditional documents, as introduced above. Digital documents require an interpreter for their bit streams in order to be meaningfully preserved (i.e., accessible). Traditional documents are directly readable by a human (implicitly assuming an educated adult human interpreter). A reader of a traditional document must of course understand its notation and language as well as any context needed to interpret it correctly, but these requirements apply equally to digital documents. The notion of an interpreter for a digital document is discussed in more detail below, but the key point is that an extra interpretation step is required to make digital documents readable by humans.[16]

---

[15] CD-ROM publications may seem to contradict this thesis, but the only relevance of the medium to these publications lies in the limitations of its capacity, access speed, transfer rate, immutability, etc., which constrain the publications that utilize this medium.

[16] The issue here is that a digital document must be interpreted (by software) in order for it to be rendered human readable. This is analogous to the fact that traditional documents must be rendered in certain ways (printed in OCR font, encoded in barcode, keypunched, etc.) in order for them to be made machine readable. The proper meaning of the term 'machine readable' is that a document is capable of being read by a computer, although the term is sometimes misleadingly used (for example, when referring to 'machine readable records') to denote information that has already been read by - and ingested into - a computer system. The terms 'machine readable' and 'human readable' should properly be seen as complementary: they denote information that is in a form that is capable of being read by a machine or a human, respectively.

Saving the bit stream of a digital document without saving an appropriate interpreter for that bit stream is analogous to saving hieroglyphics without saving a Rosetta Stone. Yet this analogy is inadequate in a crucial way: the interpreter of a digital document is an executable program, not simply another document. A program is just another bit stream, i.e., another digital document, albeit an inescapably executable one; and as we have seen, any executable document inherently depends on an interpreter. That is, the intended interpreter of an executable document is a program; but that program itself must be interpreted. Ultimately, any such sequence of programs that interpret other programs must rest on some program that executes on a computer, i.e., that is interpreted by hardware.[17]

Since a digital document is not equivalent to its physical carrier but is instead completely characterized by its bit stream – when interpreted by appropriate software – any perfect copy of a digital document is just as 'original' as any other copy. Such copies are analogous to original instances of a given edition of a printed publication rather than a unique original manuscript; yet because a digital document's bit stream requires an interpreter, this analogy is not quite correct either. It is not obvious how to define a 'digital-original' document. One option is to ban the use of the term 'original' when applied to digital documents, but this seems unnecessarily restrictive. An alternative is to define the concept in operational terms, using an analogy to the traditional concept of an original document:

> A 'digital-original document' is any representation of a digital document that has the maximum likelihood of retaining all meaningful and relevant aspects of the document.[18]

---

[17] A human can serve as the 'hardware' that interprets a program, but such 'hand execution' is notoriously error-prone and hopelessly inadequate for interpreting any but the most trivial programs.

[18] The meaning of this definition depends on what constitutes the 'maximum likelihood' of retention, as well as what constitute 'meaningful' and 'relevant' aspects of a document. The definition itself does not specify whether these criteria are to be formally defined or left as a matter of human judgment. As a first approximation, the original logical format (i.e., the 'native format') of a digital document appears to be the best candidate for a digital-original form, assuming that this native format can be kept usable in the future.

All of this argues that it is not possible to save a digital document in the same way that one saves a traditional document, simply by saving its physical carrier. Even saving the bit stream of a digital document along with the bit stream of a program that can interpret it is not enough, since executing this interpreter program ultimately requires hardware, and saving hardware in working order after it has become obsolete is infeasible. This is the crux of the problem of preserving digital documents.

3. IMPLICATIONS OF DIGITAL DOCUMENTS AS PROGRAMS

We have argued above that digital documents are essentially computer programs and require interpretation in order to be made readable by humans. A computer program is a sequence of commands (expressed in some formal language) that is intended to be interpreted by an interpreter that understands that language.[19] The computer science meaning of the term 'interpreter' used here is somewhat narrower than the common meaning of the word. The interpreter of a computer program is simply a process of some sort that knows how to perform the commands specified in the formal language in which that program is written. The term does not imply any artistic or intellectual 'interpretation' of the program; rather, it has a direct, operational meaning, i.e., the interpreter simply *does* what the program specifies.[20] Unless otherwise qualified, the term is used in this narrower sense throughout this report.

A digital document is a program, i.e., a sequence of commands in some formal language, intended to be interpreted by some interpreter that understands that language and interprets the document to 'render' it readable (or, more generally, 'usable') by a human. The interpreter can be hardware (e.g., a printer, which prints a simple text file) or software (e.g., an application program). Most non-trivial digital documents rely on software interpreters (application programs), for reasons discussed below.

If a digital document is interpreted by an application other than the one by

---

[19] Of course the 'intention' of the creator of a program may be unknown: for example, a program might be written to serve as a poem or joke. Nevertheless, the normal purpose of a program is to be interpreted by an appropriate interpreter: in this sense a program is implicitly intended to be interpreted by some interpreter that understands the language in which it is written.

[20] For example, a simple mathematical programming language might consist of commands to perform arithmetic operations on numerical quantities. A program in this language would specify a particular sequence of such operations to be performed on one or more numerical arguments; the interpreter would execute this program by performing these operations on the given arguments, producing a numerical result.

which it is intended to be interpreted, the result may or may not be correct.[21] This is equivalent to running a program on a computer whose processor is not the one on which that program was intended to be run. The outcome in both cases is highly unpredictable and unreliable: in the unlikely event that the program (or document) runs at all on the unintended interpreter, it is likely to behave erroneously – if not disastrously.

If an interpreter is implemented in software, it must in turn be interpreted in order to run. Again, the interpreter's interpreter can be either hardware or software, but any sequence of software interpreters must ultimately invoke some hardware interpreter that actually runs all of the software above it.

Note that hardware interpreters are generally restricted to fairly simple and static formal languages (i.e., fixed instruction sets) whereas software interpreters can be arbitrarily complex and are easily changed. It is generally (if arguably) easier and cheaper to change and test software than hardware, and it is generally easier and cheaper to upgrade software than it is to install new hardware, which is at least part of the reason that hardware is typically restricted to interpreting simpler, more static, and better-defined formal languages. Further, since it is almost never cost-effective to build hardware without fully specifying it in advance, its functionality is typically very well defined: this is necessary to allow hardware manufacturing facilities to build the hardware so that it works at all – let alone does what it is expected to do.[22] Software, on the other hand, need not be well specified in order to be built, since there is no physical 'manufacturing' process involved. There are many

---

[21] While it is *possible* that some other interpreter might correctly interpret the document, the correctness of its interpretation would have to be explicitly proven for any interpreter other than the intended one. Furthermore, it is difficult to conceive a basis for such proof other than direct comparison against the result of interpreting the document with its intended interpreter, since we are currently unable to capture and specify the detailed behavior of an arbitrarily complex software interpreter. If the behavior of some new, alternative interpreter could be proven equivalent to that of the intended interpreter of a document, then the new interpreter could be used in place of the intended one; but unless the two interpreters could be proven equivalent in all respects, this equivalence might have to be established on a per-document basis.

[22] The concept of a 'specification' is used throughout this report in its formal sense, i.e., meaning a complete and rigorous description of an entity, not merely a list of requirements that the entity must satisfy. In particular, it should be possible to create (or recreate) an entity from its specification.

good reasons to fully specify software – and to update this specification as the software evolves; yet in many cases, even if software is specified in advance, it is generally modified until it works as desired, at which point the modified code is used as its own specification. Replication of the modified software at this point is easily accomplished by copying the bit stream that is the software, without necessarily modifying the software's original specification. (If similar after-the-fact modifications were done to hardware, the results could not be replicated without 'reverse engineering' the modified hardware and modifying its specification accordingly so that additional instances of the modified device could be built.) Even when software is fully specified, any discrepancies between its specification and its behavior can often be ignored; such discrepancies would make it impossible to manufacture hardware.

To reiterate, we must – and generally do – develop detailed, high-quality specifications for hardware, whereas we often neglect to do so for software, despite the best intentions of software methodologists. It is arguably less important to specify software fully, since any program is already a formal, implicit specification of its own behavior (although since it is implicit, this specification may be difficult for humans to read and understand).

The need to fully specify hardware in advance – as well as the inherent cost of building complex hardware and the performance penalties that have been experienced using complex instruction set computers (CISC) as opposed to simpler, 'reduced' instruction set computers (RISC) – have kept hardware interpreters much simpler than software interpreters. The formal languages interpreted by hardware interpreters are therefore restricted to be relatively simple, whereas the languages that can be interpreted by software interpreters can be arbitrarily complex. Since digital documents utilize operations that can be quite complex (such as 'fill and justify this paragraph' or 'insert an animated graphic here') they require complex interpreters, which is why most complex digital documents rely on software interpreters (application programs) rather than hardware.

When a program 'runs' (i.e., is executed by its intended interpreter) its behavior becomes manifest. Attempting to infer a program's behavior without actually executing it is notoriously difficult and unreliable, even for the most trivial programs, since this requires the ability to infer the behavior of the interpreter (without being able to run it) and then recreating how the given program would behave when interpreted by that interpreter – all

without being able to test this inference by actually running the interpreter or the program in question.

This implies that in order to reliably recreate the behavior of any program, it is necessary to run it on its intended interpreter. This can be done either by saving its original interpreter in executable form or by saving a specification of that interpreter sufficient to allow creating another interpreter with equivalent behavior in the future. If the interpreter in question is a program, it is generally infeasible (given the current state of the art) to create an adequate specification for its behavior, so the only alternative is to save the interpreter itself. Fortunately, this is easy to do, since the interpreter (being software) is just a stream of bits: if we can save the bit stream of a digital document itself, we can save the bit stream of its interpreter program using identical methods.

On the other hand, if the interpreter of a program is hardware, it is infeasible to save it in usable condition for very long, since it will quickly become unmaintainable. We must therefore instead save a specification of a hardware interpreter that is sufficiently detailed and accurate to allow us to recreate its behavior in the future. Fortunately, this too is relatively easy to do, since we have seen that the behavior of hardware is comparatively simple and that all hardware must be fully specified in order for it to be built in the first place. Given such a specification, we might choose to recreate the physical hardware in the future, but a more attractive option is to emulate that hardware on a future computer, thereby avoiding the physical recreation of obsolete hardware whenever we want to access an old digital document.

Now, in order to save a digital document, what must we do? Since we recognize that a digital document is essentially a program intended to be interpreted by appropriate application software, we must save the bit stream of that interpreter (plus whatever software environment it requires) along with the document. But since this interpreter is a program, it must itself be interpreted by a hardware interpreter in order to run, so we must save that hardware interpreter as well. As argued above, the best way to do this is to save a specification of that hardware to allow recreating its behavior in the future, by means of emulation. Using emulation, we would treat the saved hardware specification as a program to be executed on future hardware, thereby virtually recreating the original hardware, on which we would then run the saved original application software to interpret the saved digital document.

4. THE CONCEPT OF EMULATION

The computer science term 'emulation' denotes a process in which one computer is used to reproduce the behavior of another computer with such fidelity that the emulation can be used in place of the original computer. We will call the original computer, whose behavior is being emulated, the 'emulated' system, and we will call the computer that is performing the emulation the 'emulating' system; we will call the entity that defines this process an 'emulator', and we will call the process itself 'emulation'. Emulation is a well-proven technique that has been a part of mainstream computer science for decades.[23]

Emulation is closely related to the concept of a virtual machine.[24] A virtual machine is something that serves the role of some computer that does not actually exist. Virtual machines are typically generalized or simplified analogues of real computers, implemented by software.[25] Since a virtual machine serves the role of a computer, programs can be written to run on it, even though it itself is implemented as a program.

Since an emulator is a program that makes one computer act like a different computer, an emulator can be thought of as implementing a virtual machine that corresponds to the emulated computer and runs on the emulating computer.[26] Conversely, any implementation of a virtual machine is a

---

[23] It is important to distinguish emulation from simulation: simulation is a process that shows how a system would behave but does not actually behave that way. That is, a simulation is not a replacement for a simulated system (for example, aircraft simulators do not fly), whereas an emulation can be used in place of the original system to actually accomplish whatever the original system accomplished. In this way, emulation can recreate the behavior of an obsolete computer that no longer exists.

[24] Computer scientists say that an entity is a 'virtual X' if it is not an X but can be used as an X for some purposes. The paradigmatic example is virtual memory, in which secondary storage (e.g., disk) is used to create the effect of a computer's having additional primary storage (i.e., 'RAM').

[25] Since a virtual machine is implemented by a program, that program must run on some 'host' computer, but the host computer that it runs on is not (normally) the machine that is virtually created.

[26] Emulation implements a virtual machine in the sense that it creates the effect of a machine that does not physically exist.

program that emulates that virtual machine on some host computer.[27]
The distinction between these two concepts, while somewhat subtle, consists of two related facts. First, a virtual machine does not normally correspond to any real computer, whereas an emulator typically emulates some particular (historical, existing, or proposed) computer. Second, one common design criterion for a virtual machine is that it be a simplified platform that is easy to host on multiple real platforms, whereas an emulator – being constrained by the complexity of the real computer that it emulates – may not be able to satisfy this criterion.
Although in principle either software or hardware systems can be used to emulate other systems that consist of either software or hardware, the term 'emulation' is not normally used to refer to the use of hardware to recreate the behavior of software: in such cases we simply say that such hardware 'implements' the software. However, either hardware or software can be used to emulate hardware systems, and software can be used to emulate other software systems. Small portable computers sometimes use programs to emulate the behavior of a hardware modem to avoid the need for additional hardware, while many hardware devices (including modems, processors, and terminals) emulate the behavior of other hardware devices, usually to provide standard functionality. Similarly, software is often used to emulate the behavior of other software (such as GNU's 'GhostScript' viewer for PostScript or the WINE Windows Emulator).

4.1 Emulating hardware is easier than emulating software

We have already established that preserving digital documents in their original, executable form requires preserving the application programs that know how to interpret them, the operating systems and other system software environments in which those application programs run, and the hardware on which these application and system programs run. Since it is possible to emulate either hardware or software, preservation could be based on one of three different emulation approaches. First, it would be possible to directly emulate the behavior of the application programs that render digital documents, thereby ignoring the system software and hardware that underlie

[27] A program that implements a virtual machine is an emulator in the sense that it makes the host computer act like a different computer (i.e., the virtual machine).

these applications. Alternatively, application programs could be saved as bit streams, and the system software that runs these applications could be emulated, ignoring the underlying hardware. Finally, both application and system software could be saved as bit streams, and the underlying hardware could be emulated to run all the saved software.

On the basis of the background discussion above, however, the first two of these alternatives are not practical, since it is infeasible to create specifications of software that capture its true behavior. The behavior of even the simplest program is generally too complex to describe in ways that are adequate to allow emulating it: the only reliable specification of the behavior of a program is the program itself, and that 'specification' is only understandable by the program's proper interpreter. It is therefore infeasible to emulate the behavior of obsolete software.

On the other hand, it is relatively easy to create specifications for hardware, since the behavior of hardware is generally simpler than that of software and since hardware must be well specified in order to be built in the first place (unlike software). The preferred approach, then, is to save specifications that allow emulating hardware: this emulated hardware can then be used in the future to run any programs whose bit streams have been saved.

However, there is still another choice to be made. Saved specifications for obsolete hardware can equally be used to create hardware or software emulators in the future. If performance is paramount, it may be preferable to create hardware to emulate obsolete hardware. However, when the primary motivation for using emulation is to provide a flexible way of recreating the behavior of a range of systems, software is the preferred choice, since software is easier to modify and extend than hardware.

If the motivation for using emulation is to preserve the functionality of obsolete hardware and software systems, then it is necessary to ensure that it be easier to preserve the emulating system than it would be to preserve the emulated system. As argued above, obsolete programs can be preserved simply by saving their bit streams, whereas obsolete hardware is most easily preserved by saving a specification of the hardware, rather than attempting to preserve the physical hardware itself. And while it is possible to save detailed, implementation-level specifications that would allow reimplementing obsolete physical hardware in the future, it is easier to save high-level, logical specifications to allow emulating the behavior of that hardware in future software.

In fact, for preservation purposes, it should be sufficient to emulate the logical behavior of an obsolete computer system at a relatively abstract level. Properly designed application software does not depend on timing or other internal details of processor operation but rather on the logical behavior of processor instructions, which can be specified in a relatively simple manner.[28] This makes the creation of hardware specifications that can enable future emulation even easier, since these specifications can be at the abstract level of logical instructions and operations.

It may seem unwarranted to assume that future hardware will be able to emulate any previous (obsolete) hardware, but this assumption actually rests on quite firm ground. The logical operations performed by current digital computers are instances of a well-defined class of mathematical computations known (suggestively) as 'computable functions'. The basic instructions executed by such computers are generally defined in terms of the simplest logical and arithmetic operations. Whatever kinds of unimaginable operations future computers perform (such as quantum, multi-state computations), it is almost inconceivable that they will be incapable of performing the simple logical operations that constitute the instruction sets of current (and past) computers. It therefore seems safe to assume that any conceivable future general-purpose computer will be able to perform the computations necessary to emulate any current computer.

4.2 Relevant traditional uses of emulation

Emulation has been used in computer science for a number of purposes. One of the most common uses has been to provide 'backward compatibility'

---

[28] In general, digital devices are designed to have very well-defined logical behavior. For example, all of the complexity of the circuitry that implements a processor is intended to disappear at the functional level, where the processor simply performs clearly-defined logical operations. Furthermore, most of the internal optimizations and techniques used to make processors more efficient (such as the use of pipelines, caches, etc.) are designed to be transparent at the logical level; in fact, great ingenuity is employed to ensure that a processor's logical behavior would be the same with or without these techniques. Programs whose behavior relies on the internal workings or timing of a processor are generally considered erroneous, since they cannot be expected to execute properly on upgraded versions of the processor or alternative implementations.

between new computers and old ones, to allow running old software. This is directly analogous to the use of emulation for preservation: new processors have often been provided with emulators of previous processors, so that old programs could still be run without having to be rewritten. Large investments in an organization's old software would have been lost if every new generation of computer had required rewriting all of these old programs. Emulation of an obsolete computer allows an organization to continue to run its old software, which would otherwise also become obsolete. Emulation of this kind has often been provided by hardware producers as a way of allowing their customers to upgrade to new hardware without making their old software obsolete. Such emulations have been implemented in software, hardware, and everything in between, including 'microcoded firmware' that relies on system software to make new hardware appear to function like old hardware. IBM provided emulation modes of this kind in its System 360 for previous 7090 and older systems, to give customers a gentler upgrade path. More recently, Apple Computer provided an emulator for the Motorola 68000 processor when it upgraded its systems to the newer PowerPC (PPC): many application programs and even significant portions of Apple's own operating system continued to utilize older, 68000-based code running under this emulator for years.

A second common use of emulation has been in the design of new computers. The design of a new processor is typically tested before any hardware is built, by writing a software emulator that behaves the way the hardware would behave. This allows exercising, verifying, and refining the design without building any hardware, as well as beginning to develop operating system and application software for the new processor before it is manufactured (or even prototyped). For this purpose, high-level, logical emulators may be written to allow verifying the completeness and utility of the new processor's instruction set, whereas low-level, more detailed emulators may be used to verify the internal workings of the processor, such as timing issues.

In addition, emulation has been used to gain historical perspective on the development of computers, such as by emulating the earliest computers. For example, an emulation of the EDSAC computer, designed and implemented at the Cambridge University Mathematical Laboratory under Maurice Wilkes in 1949 has been used to help understand the earliest instance of debugging, as experienced by Wilkes when writing a program to calculate Airy's integral,

the solution to the second-order differential equation y" = xy. An original paper tape copy of Wilkes' 1949 program was discovered in 1979 and was run on an ESCAC emulator program, written for an Apple Macintosh computer in 1992.[29] Similarly, a Java applet emulator for the very first computer, Charles Babbage's Analytical Engine, designed in 1842 but not built at the time, can be found on the web.[30]

Finally, an interesting use of emulation has arisen in recent years that is of particular relevance to preservation. A 'retro-computing' community has developed emulators of obsolete video and computer game machines that run on modern computers. These emulators, which are freely available on the net, allow their users to run obsolete game software on modern computers, under emulation. This is analogous to using emulation to render digital documents that depend on obsolete computers. The experience of this community is particularly interesting because computer games are notorious for placing heavy demands on their supporting hardware and on being highly dependent on the idiosyncrasies of this hardware, including interface devices, displays, execution and interaction timing, etc. The fact that obsolete games are routinely and successfully run under emulation suggests that emulation is capable of addressing any hardware or interface dependencies that may be exhibited by digital documents.

---

[29] See Campbell-Kelly, M., 'The Airy tape: An early chapter in the history of debugging', *IEEE Annals of the History of Computing*, 1992, vol. 14 (No. 4), pp. 16-26.

[30] At http://www.fourmilab.ch/babbage/applet.html.

5. USING EMULATION TO PRESERVE DIGITAL DOCUMENTS

As argued above, emulation could be used to preserve digital documents by providing a way to recreate the behavior of obsolete computing platforms in the future. Emulating these obsolete platforms would allow obsolete application software to be run, which would in turn allow rendering saved digital documents as they were originally rendered. The following explains what this approach would accomplish and how it might work.
Since all digital documents are inherently software dependent, they are essentially programs, which are generally intended to be interpreted by other (application) programs, which are in turn intended to be run within a system software environment on some underlying hardware platform. In order to save a digital-original document (in the sense defined above) it is necessary to save the entire software environment that was originally used to interpret that document in its native format and to be able to execute that software in the future, so as to recreate as much as possible of the original behavior of the digital document. In principle, it is straightforward to save a digital document and all of its required software, simply by saving the bit streams of these entities (though keeping these bit streams readable requires continually copying them to new media as older media become unreadable or obsolete). However, saving such bit streams in executable form is more problematic, since such execution requires preserving obsolete hardware in some usable form. Physically preserving such hardware for more than a decade or two is quite impractical, but emulation provides a virtual way of doing this.

5.1 What would emulation achieve?

Successful emulation would allow saved digital documents to be rendered on future computers in essentially the same way they were rendered on their original computers to their original audiences. A future user would sit at some future computer and interact with an emulation of an obsolete computer running obsolete software rendering saved documents in as close to their original form as possible. This would be analogous to hearing an ancient performance of a Gregorian chant on a modern sound system (though

emulation would capture more of the behavior of a digital-original than this analogy suggests).[31]

Emulation would effectively preserve a digital-original that would be somewhat analogous to an ancient book or manuscript – except that it would not be fragile, unique, or irreplaceable and could be accessed from any computer. As is the case for an ancient document, however, reading this digital-original would require some historical skills and context on the part of the reader. In particular, emulation of an obsolete computer would recreate a machine that the reader might have to understand in order to run the document's original software; and running that obsolete software might require unusual knowledge and skills pertaining to its obsolete interface. For example, suppose that future computers evolve to be voice-driven, rather than keyboard-and-pointer based; running a 1990s era word processor on such a machine would require using a menu or command style of interaction that would no longer be in common use. Someone attempting to read such a document in its original form might require considerable help in figuring out how to use the ancient software and the emulated computer on which it runs, and it is unlikely that there would be some helpful guru down the hall with expertise of this kind in obsolete systems.[32] Nor would there be handy user manuals available for the systems in question, except perhaps in online form-which might present the same problems as reading the document itself.[33] Any telephone numbers or network or e-mail addresses for technical support of the obsolete systems would likewise have long since ceased to function.

One solution to such problems would be for the emulating system (which might be provided by a library) to provide its own online help – in a form familiar to the future user – that would enable an untrained user to access and use the virtual obsolete computing environment containing the digital document in question. For users who are less concerned with seeing an old

---

[31] A better analogy might be a modern recreation of a Renaissance edition of Euclid's Elements that
reproduces the functioning three-dimensional geometric 'pop-up' figures that appeared in such editions.

[32] This is analogous to the challenge facing a modern reader of an ancient manuscript.

[33] However, migration could be used to convert such documentation into contemporary form; this would not violate the authenticity of the document itself, since only the documentation about how to read the document would be converted.

digital document in its original form or who have less time and effort to invest in learning to use the obsolete system required to access that form, an alternative solution would be to extract a contemporary 'use copy' (i.e., a 'vernacular' version) of an old digital document from the emulated digital-original. The emulation environment would have to provide 'hooks' to allow future programs to access the content of the emulated digital-original so that they could perform such extraction. At a minimum, these 'extraction hooks' should permit (1) some form of 'view' extraction (i.e., the ability for the user to save some rendition in a current vernacular form); (2) some kind of 'content' extraction (i.e., the ability to save the core content of a document, whether text, imagery, sound, animation, etc., in some appropriate vernacular form); and (3) some kind of search capability (i.e., the ability for the user to enable future applications to search across preserved digital documents without emulating and rendering them).

5.2 Emulation versus migration

Extracting vernacular use-copies from digital-originals bears some resemblance to migration: in both cases, one logical format is converted into another, with potential loss of content, form, structure, behavior, and meaning. However, there are several important advantages to performing this process by extraction from a digital-original rather than migrating the original into successive new forms over time. For one thing, migration entails the loss of the digital-original, since the assumption is that migration is performed whenever it appears that the current form of a digital document (whether the original or some later, migrated form) is about to become obsolete. Unless emulation (or some other, as yet undiscovered, approach) is used to preserve the digital-original in executable form, it is effectively lost (whether or not it is physically discarded) once it becomes obsolete. Migration therefore retains no usable preservation copy, whereas the emulation scheme described here retains the digital-original as a preservation copy.

In addition, migration is performed successively, from each migrated form of a document to the next, resulting in a potential accumulation of conversion errors and increased likelihood of corruption. Once any conversion in any migration cycle has corrupted a document, that corruption cannot be undone, since all previous versions of the document (including the original)

are unusable.[34] On the other hand, by retaining the digital-original preservation copy of a document, emulation allows each extraction to be made directly from this digital-original. Therefore even if one such extraction results in a corrupted version of the document, the next extraction (into the next required vernacular form) will be unaffected by the corrupted conversion of the previous extraction, and conversion errors will not accumulate over time, since no successive conversion is performed.

Furthermore, since migration retains no usable digital-original, it is impossible for it to correct – or even detect – any cumulative corruption that may be introduced by its repeated conversion of a document into successive logical formats, since there is no original against which to compare later, migrated versions of a document.

Finally, migration must be done while the current form of a digital document is still readable; once it becomes unusable, it can no longer be converted into the next migratory form. The temporal window for performing migration may therefore be quite narrow, since it cannot be done until a new migration form is defined and standardized, but it must be done before the previous migration form has become obsolete.[35] Similarly, the frequency with which

---

[34] The informal notion of a migration 'cycle' is used here to mean a single conversion effort involving some set of documents that would become unusable if not migrated during that cycle. In practice, it is not always easy to define what constitutes a given logical format or when it becomes obsolete (thereby requiring a new migration cycle). For example, new versions of word processors or graphics programs may be incompatible with older versions, requiring a migration cycle to convert documents created by those older versions before they become unusable – despite the fact that the new version of the program perpetuates (a new version of) the 'same' logical format.

[35] Note that a similar temporal window applies to validating the authenticity of emulation. In order to demonstrate the ability of a given emulator to reproduce the authentic behavior of a particular saved document, the behavior of that document under emulation would have to be compared with its behavior on its original hardware before that hardware becomes unusable. However, since emulation saves a document's original software, the temporal window in this case is not defined by software obsolescence but only by hardware obsolescence; and while some programs last longer than some hardware platforms, most hardware platforms outlast many programs. The average temporal window available for validating emulation should therefore be larger than the average window for performing migration, since many

migration must be performed is not under the control of the preservationist: it is determined by the obsolescence of logical document formats, software, and hardware. Current indications are that such migration cycles may have to be as frequent as every five years. Furthermore, if one such migration is missed for a given document, that document is likely to be lost forever, since it will become inaccessible once its previous form becomes obsolete. On the other hand, emulation allows extraction to be performed at any time in the future, with no urgency, since the digital-original preservation copy is always retained in readable form. Any number of potential extraction cycles can be skipped with impunity, as they might be, for example, for documents that are rarely accessed or are accessed mainly by scholars interested in their digital-original forms rather than in vernacular use-copies.

In addition, a crucial aspect of migration is that *all* digital documents that are to be preserved must migrate during *every* migration cycle: that is, *every* document must be processed to be converted into the next migration format. Any document that is not converted during a given migration cycle is likely to be lost. Yet most documents in a given corpus are rarely accessed, especially within the time between successive migration cycles. So most of the effort of migrating a corpus in a given cycle will in a sense be wasted, since no one will access the resulting converted form of most documents in the corpus; the only reason to convert most documents is to avoid losing them, i.e., to convert them into a form that will still be readable when the next migration cycle must be performed. Moreover, the processing that must be performed on each type of digital document will be different: different logical formats –

---

logical formats will become obsolete in the time it takes an average hardware platform to become obsolete. Moreover, once an emulator has been shown to reproduce the behavior of the original system it is emulating, no further effort should be required to demonstrate that the behavior of any document – relying on any software that ran on that original system – should be faithfully reproduced by the emulator. Although individual documents (and the application programs that interpret them) can exhibit a tremendous range of potential behavior, they do so by combining a relatively small number of elementary operations provided by the hardware on which they run. In principle, an emulator need only reproduce these elementary operations correctly in order to guarantee that any combination of those operations will be reproduced correctly. Nevertheless, validating emulation in this way remains an open issue, requiring further research.

and possibly even different kinds of documents having the same logical format – may require different conversion at each migration cycle. This makes migration processing even more expensive. In contrast, retaining the digital-original preservation copy of a digital document for emulation purposes requires no processing whatsoever, except and until the document is viewed or extracted into a vernacular form. All document types and formats are treated identically, since all are simply considered bit streams that are preserved without change over time.

It must be kept in mind that the bit streams of digital documents, whether converted by migration or retained in their digital-original forms for emulation, must be preserved if the documents are not to be lost. This requires copying the bit streams of all documents to new storage media as older media become obsolete.[36] This must be done regardless of whether migration or emulation is used to address the problem of logical format obsolescence. In the case of migration, however, there is an additional problem. It might be tempting to think that bit streams could be copied to new media at the same time as the migration process is performed, but this is unlikely to be feasible. Storage media and logical formats become obsolete at independent rates, so attempting to perform both processes at once would result in either performing one process more often than necessary (at considerable expense) or losing documents by delaying one process until it is time to perform the other (i.e., neglecting to either copy or convert information before it becomes unusable). The only feasible approach is to perform each process whenever necessary, but this may create a data management nightmare involving conflicts between the competing needs – for copying bit streams and converting logical formats – of overlapping sets of documents.

Finally, it might be argued that emulation would require more processing than migration when a digital document is actually rendered, but this is unlikely to be the case. Each approach requires running some program. In the case of migration, this would always be some current application program that can render whatever is the current migrated form of the document, running within the current system environment of a current computer. In

---

[36] Unfortunately, the term 'migration' is often used to refer to this process of copying bit streams to new storage media, as well as the conversion of logical formats. We use the term 'migration' to mean logical format conversion throughout this report.

contrast, under emulation, the document's original application would be run within its original system environment under an emulator on a current computer. Since each approach simply requires running a different program on the same computer, they would appear to have similar processing requirements. And while emulating a given computer is less efficient than running that same computer, emulating an obsolete computer on a future computer typically results in better performance than the original computer delivered, because newer computers are generally faster than older ones. Executing an old application under emulation on a future computer therefore often results in better performance than that program originally exhibited. Conversely, a new application often taxes the performance of its computer to the limit, since it fully utilizes the system's resources. Since migration relies on the use of new (current) applications at all times, it may therefore be more computationally expensive than emulation.[37]

[37] The emulation scheme does require the creation of an emulator specification for each obsolete computing platform and the porting of an emulation environment onto new platforms, but this would ideally be a one-time cost for each such platform, as discussed below. The actual cost of running such emulators is unlikely to approach the cost of running future applications as required by migration.

6. How would emulation work?

So far, we have been rather abstract about how preservation would actually be performed using emulation. The following presents a high-level technical overview of what would be required to make this approach work. A later section offers a 'concept of operations' for how libraries would manage and use an emulation-based preservation and access scheme on a day-to-day basis. Each digital document to be preserved by an emulation scheme will in general consist of some set of digital objects, such as text or multimedia files, directories, database records, or other entities, depending on the system involved. We have so far spoken rather loosely about such documents as being represented by a bit stream. In practice, the representation of a given digital document will consist of a collection of digital entities, structured in some system-dependent way that is understood by the application and system software that is intended to read the document. Generally speaking, the entire bit stream representing this collection of entities must be saved in its native form, i.e., so as to be accessible and readable by its associated software. The precise mechanisms required to make this work in the general case have yet to be defined, but in principle, they involve encapsulation[38] of the bit stream corresponding to a document along with the bit streams of the software components needed to access that document, with sufficient descriptive metadata to allow reconstructing an environment in which those bit streams have the required relationships to each other to permit the software to access the document. Such mechanisms are not trivial but fall within the realm of standard computer science techniques.

A simple approach to storing the bit streams of a digital document with the application and system software needed to access and interpret it would be to capture an 'image' of the installed software suite along with an 'image' of the stored files or other entities that represent the document itself. The term 'image' here does not mean a digitized picture but rather a logical 'snapshot' of the bits in a computer's internal memory or storage device at a given

[38] Encapsulation is a key concept in computer science, corresponding to the notion of enclosing a collection of items in a capsule. In the current context, it implies that the appropriate bit streams and metadata would be enclosed in a logical container so that they do not become separated from one another.

moment in time, for example, when all necessary software for a given document is installed and running. Such memory images represent the state of a computer's main memory, disk drives, etc. with a specific collection of software and documents installed.

Conceptually, we could install the application needed to render the digital document in question along with all required system software and save the exact state of the computer's memory as a 'memory image' once this software is running. Restoring the memory of any identical computer to this exact state in the future would allow running this same software suite again. This memory image is a single bit stream, comprising the contents of the computer's memory at a moment in time. Similarly, the contents of a storage device, such as a disk drive, can be captured with a desired digital document installed on it; the relevant parts of the device's memory (including any directory information needed to access the document on the device) can again be thought of as single bit stream. If this 'memory image' is saved and later restored to any identical device, the software running on this computer will be able to access and render the digital document just as it did originally. Once these memory images are saved, they can in fact be restored to an emulated (virtual) computer and storage device rather than the real ones, with the same effect. This simple approach has one significant flaw, however. By combining the specific application needed to access and render a specific digital document with the system software that it requires to run, we produce a single memory image that would be utilized by each digital document that requires this application. But other documents require other applications, while most applications require most or all of the same system software. If we combine the system software with each different application, then each such memory image will include the system software, which is highly redundant. It would therefore be better to treat the common system software as a separate, common memory image, with each application program represented by a different image (along with any special-purpose system software it requires). Then to use a given application, we would restore both the common system software memory image and the specific application memory image to create the required bit stream.[39]

---

[39] This is still somewhat simplified, but it is hoped that most readers will not be bothered by the missing details, while readers who are information technologists will find this presentation sufficient to suggest what those details might be.

6.1 Creating emulators for future computers

Now, the only missing piece required to execute this saved software on the saved digital document in the future is the hardware needed to run it. Here we must facilitate the creation of an emulator that emulates the original computing platform on some future computing platform. Given such an emulator, we can run the original software for the digital document on a future computer, thereby rendering the document. But what do we need to do to create such an emulator?

6.1.1 The layered emulation approach

The most straightforward alternative is simply to write a program that emulates the original platform. If we write this program in a programming language that is usable on some future platform, then we will be able to run that emulator on that future platform.[40] As soon as a new platform is introduced that threatens to make some current platform obsolete, we would write a program to emulate the current platform on that new platform. It is doubtful whether any current programming language will last long enough to run on platforms beyond the immediate future, but all we need to do to ensure that the current platform remains available via emulation indefinitely is to generate a single executable version of our emulator of the current platform that runs on the platform that replaces it. Once such an executable emulator exists, it can be run under successive layers of emulation on successive generations of future platforms. For example, when the new platform itself threatens to become obsolete, we can write an emulator of it that runs on its successor: that emulator will allow running all of the new platform's programs on its successor, including the previous emulator of our current platform.[41] This 'layered emulation' approach requires no new

---

[40] To be usable on some future platform, the language must merely be able to generate an executable program that runs on that platform. If the language is compiled, its compiler need not run on the future platform – it must merely generate code for that platform (however, if the language is interpreted, an interpreter for it must run on the future platform).

[41] As soon as the original emulator program has generated an executable emulator, the 'source' program can be discarded, since the executable emulator will continue to run indefinitely on future platforms under successive layers of emulation. It is thereafter irrelevant whether the language in which the emulator program was written can be

technology and could therefore be undertaken immediately: it simply requires a specific coding effort to write a new program that emulates each platform and runs on its successor. This approach has two disadvantages, however. First, writing a complete emulator program for each platform is a non-trivial task. Second, running emulators under successive layers of emulation – one for each generation of future platform – would be relatively inefficient.

6.1.2 The emulator specification approach

An alternative to layered emulation is to write a specification of our current computing platform that is sufficient to allow new emulator programs to be written (or generated automatically) in the future, based on that specification. This 'emulator specification' approach frees us from dependence on normal programming languages as well as addressing the two disadvantages noted above: specifications should be easier to write than complete emulator programs, and such specifications should allow emulators of current platforms to be generated for successive future platforms, rather than relying on multiple layers of emulation.[42] Yet unless we can automatically generate emulators from these specifications, we would still have to write specific emulator programs of each obsolete platform: we have simply shifted the burden for doing this from the present to the future, since saving specifications makes it unnecessary to write the emulator program of a given platform while it is still operational.[43] In fact, unless we can automatically generate emulators from specifications, this approach greatly increases the amount of work required, since it encourages the generation of one emulator of a given platform for each future platform.

---

compiled or interpreted on future platforms, since the source code of that program need never be used again.

[42] This assumes that an emulator specification of a current platform will remain understandable far enough into the future to allow it to be used to generate new emulators (of the current platform) for each new generation of future platform as it appears. However, if and when the specification ceases to be understandable, the last executable emulator that was generated from it can simply be used from then on, as in the layered emulation approach.

[43] Since this approach assumes that we will continue to create new emulators of current platforms for successive future platforms, it relies on the fact that this can be done using the specification of a platform long after it has become obsolete.

It we take this approach, then every time we develop a new computing platform, since it will eventually become obsolete, we would also create an emulator specification of that platform, which we would save for future preservation purposes. If the recognition of the need for such emulator specifications becomes sufficiently widespread, computer vendors should recognize a market opportunity and begin to create such specifications, which they are ideally situated to produce, since they presumably understand their own computers well enough to specify them quite easily. Otherwise, some other market segment would have to provide such specifications; but whoever produced them, they would become a new class of product, to be sold to libraries, archives, and recordkeeping organizations.

6.1.3 The specification interpreter approach

Although the emulator specification approach has advantages over layered emulation, it requires a new emulator program to be developed from the saved emulator specification for a given platform every time a new future computing platform is developed on which it is desired to access saved digital documents that depended on the original platform. A more efficient approach would be to define a standard language for emulator specifications that can be interpreted by a general-purpose emulation specification interpreter, which is a program that would run on future platforms. Then rather than having to implement each individual emulator on each new computer, we would simply have to implement the single emulation specification interpreter on each new computer (i.e., 'port' it to each new platform), and it would interpret all previous emulator specifications. This 'specification interpreter' approach would effectively implement all previous emulators at once for a given new platform.

6.1.4 The emulation virtual machine approach

Taking this one step further, it should be possible to implement the emulation specification interpreter as a program that runs in a simple 'emulation virtual machine' (such as the Java Virtual Machine, though ideally simpler to implement). In order to port the emulator specification interpreter to a new platform, it would then be necessary merely to port the underlying emulation virtual machine to that new platform. The virtual machine concept is a proven computer science technique: a virtual machine can

typically be ported to a new platform by writing a relatively small number of 'primitive' program modules for the new platform.

This 'emulation virtual machine' approach would require choosing or defining a standard emulator specification language, along with a standard emulation virtual machine formalism, and writing an emulator specification interpreter for that virtual machine. Although this suite of interrelated standards would itself not be expected to last forever, it would not be necessary to translate emulator specifications into a new language if the original language were superseded. Instead, whenever a new emulator specification language is developed in the future, a new emulator specification interpreter would be developed for it, to run on whatever is the current emulation virtual machine. All previous emulator specification interpreters would be retained to interpret older emulator specifications written in the previous languages. Whenever the emulation virtual machine itself is redefined, an emulator specification of it would be developed in the current emulator specification language: this would allow the new emulation virtual machine to emulate the previous emulation virtual machine (as in layered emulation).[44] In this way, any new emulation virtual machine will automatically run all previous emulators with no additional effort. Though this may sound like magic, it is in fact based on standard, proven computer science techniques.[45]

---

[44] An emulation virtual machine is itself a (virtual) computing platform, which can threfore be specified by an emulator specification just like any real computing platform.

[45] Note: it has been suggested that emulation simply involves the migration of software rather than documents. But emulation retains the original software for a document intact: it is not modified in any way (which is what migration normally implies). It is in a sense 're-hosted' from its original platform onto an emulated platform, but the whole point of emulation is that this avoids the conversion involved in normal re-hosting. And once the original software runs under emulation, it is never re-hosted again. Similarly, the emulation specification interpreter never migrates, since it runs on a single (virtual) platform (i.e., the emulation virtual machine). The virtual machine itself might be said to migrate onto successive hardware platforms, but this is a dubious use of the term migration, since the whole point of a virtual machine is to allow it to be hosted on multiple platforms without modifying it in any way. In any case, even if reimplementing the virtual machine is considered an instance of migration, this should require orders of magnitude less

In principle, only the most recent emulation virtual machine specification need be retained in human readable (i.e., 'vernacular') form,[46] to allow implementing new versions of the emulation virtual machine on new platforms; previous emulation virtual machine specifications need not be human readable since they can be run (under emulation) on the current emulation virtual machine.[47] Note that an emulation virtual machine specification can be preserved in human readable form just like any other digital document, by encapsulating its bit stream with that of any convenient viewer program, along with an emulator specification for a hardware platform capable of running that viewer.

6.2 Hardware configurations, versions, and modular emulation

In many cases, application software depends merely on the processor and main input/output devices of a computing platform, and emulating this hardware should be sufficient to run such applications. However, some applications make use of special auxiliary chips, co-processors, graphics, sound, or video cards or boards, or additional peripheral devices. In some cases, applications may also have special configuration requirements, such as additional graphics memory, special disk partitions, etc. Many such cases may be handled by emulating a generalized, 'robust' standard configuration of a given platform, that includes all of the devices, options, and memory that were available for that platform: applications that do not require all of the resources available in this configuration should run in it anyway. However, there may be cases where it is necessary to run specialized configurations. Similarly, different models of processors or computers often have different features or capabilities. Families of processors often define a common 'core' instruction set that may be extended in some models. While it may be possible to define a generalized 'robust' standard exemplar for such families or

---

effort (and be incomparably less error-prone) than individually migrating every document into a new logical format whenever its current format becomes obsolete.

[46] That is, in a form that can be read and understood by contemporary human readers.

[47] Nevertheless, it might be safer to preserve the specifications of all previous emulation virtual machines in human readable form to allow resolving any questions about whether the use of layered emulation virtual machines is faithfully interpreting emulator specifications written in older emulator specification languages.

collections of models, there may be cases where specific versions of hardware must be emulated.

A general solution to this problem would be to make the emulation environment capable of modular emulation. That is, individual devices would be emulated by individual modules that represent those devices' options and interfaces to other devices, in such a way as to allow plugging emulation modules together the way the original devices were plugged together. Similarly, the core instruction set of a family of processors could be implemented as a single emulation module, with extended capabilities being supplied by additional emulation modules. Configuration information would then allow automatically creating customized emulated systems, as required.

7. UNRESOLVED QUESTIONS AND ISSUES

The foregoing technical discussion omits a number of details that would be necessary to make emulation-based preservation work. For example, encapsulating digital documents with their software and associated hardware emulator specifications requires appropriate metadata to explain how to open the encapsulation and access its various pieces to render a digital document. Such explanatory metadata must be made more easily readable than the encapsulated digital documents themselves, so that a user need not open the encapsulation (i.e., using emulation) in order to read the explanation of how to open the encapsulation.

Furthermore, though an encapsulated digital document would logically include all relevant application and system software, it would be highly redundant to duplicate all of this common software with each document that needed it. In practice, it would be preferable to store common software and hardware emulation configurations in a central repository and point to the required modules in this repository from within each encapsulated document. Depending on their perception of the relevant risks and costs, individual organizations could decide whether to trust remote repositories or keep their own copies of such common digital components.

In addition, there are still a number of questions that must be answered in order to develop a viable emulation-based approach to preserving digital documents. Some of these questions are technical, while others are related to legal or social policy issues.

Some of the open technical questions are suggested or implied above. For example, it remains to be seen whether large numbers of distinct models, versions and configurations of hardware platforms must be described individually or whether generic versions can be emulated. In addition, there is the question of how future users can be helped to use obsolete systems. Migrated or extracted vernacular versions of user manuals or original online 'help' information might be provided, or future expert system 'helper' programs might be developed to help users operate obsolete hardware and software.

Perhaps the most fundamental remaining issue is whether it is realistic to develop emulator specifications that describe hardware platforms and

environments in sufficient detail to enable digital documents to be rendered authentically on future platforms. In particular, in order to capture the look-and-feel of digital documents, an emulator specification might have to capture information about such aspects of a computing platform and environment as screen resolution, color calibration, refresh rate, dimensionality, processor and storage device speed, keyboard layout, pointer device modality, sound quality, etc.

Of only slightly less concern is the fact that emulation requires the bit streams of both digital documents and their associated software to be preserved with near perfection. Programs are generally quite intolerant of incorrect bits in their bit streams, while each time a bit stream is copied onto a new storage medium (or even when it is left to sit on the same medium over time), there is a finite chance of losing some of its bits. There are technical approaches to mitigating this risk, for example using redundant storage, error-correcting codes, verification techniques when copying, etc., but the risk can never be entirely eliminated. At worst, this might result in the occasional loss of digital documents, analogous to the occasional loss of some traditional documents over time, despite our best efforts to preserve them.

Finally, there are a number of thorny non-technical questions, such as the intellectual property issues surrounding the preservation and future use of software and hardware specifications, despite their being obsolete.

8. CONCEPT OF OPERATIONS FOR EMULATION-BASED PRESERVATION

The following describes what would be involved in the day-to-day operation of an emulation-based preservation approach. For simplicity, this discussion assumes that the 'emulation virtual machine' approach (described in Section 6.1.4) has been adopted. At the end of each of the follow sections, a subsection 'Contrast with Migration' briefly indicates how these activities differ from those required under a standards/migration based approach to preservation, since migration is the most commonly proposed alternative to emulation.[48]

8.1 Initial production of digital documents to be preserved

One of the hallmarks of the emulation approach is that it allows preserving digital documents produced by essentially any software that runs on essentially any hardware. From the perspective of an organization or individual involved in producing digital documents, only two conditions must be met to ensure that a document can be preserved by some internal or external preservation agency. First, it must be legally and technically possible to supply that agency with a memory image of an installed configuration of the application and software environment required to render the document. If this environment consists of standard modules that are likely to be preserved elsewhere (e.g., standard operating system or application programs) then it is preferable to supply solely those additional software modules (if there are any) that are required to render the document in question, along with suitable metadata describing all required standard software, using standard nomenclature and designations. If, on the other hand, the software environment is nonstandard or otherwise unavailable to the preservation agency, then a complete bit stream representing the memory image of the installed software environment must be supplied.

---

[48] It must be kept in mind (as mentioned in note 5 above) that neither emulation nor migration has yet been demonstrated to be a viable way of preserving digital documents.

Second, an emulator specification must be available for the hardware environment needed to run the software required to render the document to be preserved. If the hardware environment in question is a standard configuration of a standard platform for which a standard emulator specification already exists, then it is sufficient to provide metadata denoting that specification. On the other hand, if any or all of the hardware environment is nonstandard, then emulator specifications for all nonstandard aspects of the environment must be supplied. If such specifications are not available from the hardware suppliers, they must be developed, verified, and expressed in the formal emulator specification language currently in use.

In the majority of cases, the application software, software environment, hardware platform, and software and hardware configurations will all be standard, and the software and emulator specifications for the hardware platform will all be readily available to the preservation agency. In such cases, the producer of a document need only encapsulate the bit stream of the digital document with appropriate metadata describing its required software and hardware environment, and supply this to the preservation agency. In other cases, the producer may have to encapsulate additional software modules or emulator specifications with the document. Ideally, the preservation agency should provide a 'validation facility' over the network, to allow document producers (such as publishers) to try out their submissions in the agency's actual preservation context. This validation facility would allow document producers to use their encapsulated documents and metadata to try to configure working 'preservation' copies of those documents, collecting any software modules or emulator specifications denoted by the metadata and building an executable, emulated rendering of the documents in question. Using this validation facility, a document provider would be able to verify that any submitted documents could be rendered under emulation (albeit on a current rather than future computing platform) and therefore (in all likelihood) that they would be properly preserved in the future.

Although a document producer might not be able to afford the time or effort to test every document submitted to the preservation agency using this validation facility, it could certainly do this for selected examples or a random sample. Since most producing organizations probably produce documents of a relatively small number of types (e.g., using a small set of applications) it should be feasible to test occasional examples of each primary type to verify that the preservation process is likely to work.

Additional metadata would be provided by the document producer in accord with established requirements.

8.1.1 Contrast with Migration

Most standards/migration based proposals for digital preservation restrict the acceptance of documents at ingest to a relatively small number of acceptable formats. If a document is already in one of these formats, the producer need do nothing but package the document with appropriate metadata, as above. However, if a document is in some nonstandard format, the producer may have to convert it into a different format in order for it be accepted for preservation. No validation testing is generally proposed for this approach, but it would in any case apply only to the accessibility of the document during its initial tenure in a repository, since it is impossible to test how a document would be preserved after migration (i.e., conversion into some as yet unknown future format). This inability to preview a document's future form is a key difference between migration and emulation; while this inability reduces the work a producer must do, it does so at the cost of any assurance that the document will in fact be preserved effectively.

8.2 Acquisition (ingestion) of documents for preservation

One of the key advantages of the emulation approach for a preservation agency is that it need not perform any conversion on ingestion (or at any other time). Nevertheless, it behooves the agency to perform some kind of acceptance testing, if only to verify that all required elements are provided. This may be a simple, automated check that encapsulated documents have appropriate components. If desired, the agency may also choose to do some validation testing of its own, perhaps on a random sample of ingested documents. This would be analogous to the validation test recommended for document producers above. If it does not perform such testing itself, the policy of the preservation agency should probably be to place the burden for such validation on document producers, though it should provide the validation facility to enable producers to perform such testing in a standard, controlled environment.

It is assumed here that the burden of identifying or supplying emulator specifications will rest with the document producer. However, the preservation agency may serve as a solicitor, developer, clearing house, or

repository of standard emulator specifications, as well as a repository of standard software configuration suites.
Appropriate metadata must of course be created for any documents that are to be preserved, but aside from the emulation-specific technical preservation metadata discussed above, the question of which metadata should be produced by document producers versus preservation agencies is independent of the use of emulation, and so is outside the scope of this report.

8.2.1 Contrast with Migration

In many standards/migration based proposals, ingestion involves a potential conversion step, wherein documents are converted into one of a few acceptable initial preservation formats. This would require considerable computation and potentially considerable 'hand' processing to deal with anomalies. If producers are forced to perform this conversion instead, this effort is off-loaded to them, as discussed above. In either case, there is a chance that a given document will be rejected for acceptance to the preservation process due to its being in an unacceptable form; however, conversely, documents that are accepted cannot be tested to see if they will be appropriately preserved after their first (let alone subsequent) conversion.

8.3 Ongoing preservation activities

In order to preserve digital documents using emulation, a preservation agency need (as a minimum) do the following:

1) Ensure that the bit streams of digital documents and software entrusted to its care are copied without corruption to new storage media as needed to keep them readable and accessible. (Note: this is required by any digital preservation scheme.)

2) Ensure that the explanatory metadata associated with encapsulated digital documents remains understandable. This requires converting at least the topmost level of such explanations (sufficient to explain how to read lower levels of explanation) into successive 'explanation standards' as previous such standards become obsolete. Ideally, any such conversions should be 'reversible' as discussed below under research issues. Note that this is the only step that involves processing individual documents, and this

processing applies only to the metadata for such documents, not to the documents themselves.

3) Maintain the linkages between named standard software and emulator specification components and configurations in the metadata in encapsulated documents and the stored forms of those entities in its own or remote repositories, as well as maintaining similar linkages between metadata references to named emulator specification languages, emulator specification interpreters, emulator specifications, and emulation virtual machine specifications. Linkage information of this sort should be maintained by the use of tables or indices, rather than by modifying stored names, to avoid having to modify encapsulated documents simply in order to change the denotations of any names contained within their metadata. Note that this task implies, as a side-effect, that any components that are required by any documents in the agency's repository but which are unavailable must either be located and linked to or acquired (or developed) and stored in the agency's repository.

4) Retain the bit streams of the emulator specification interpreters for all emulator specification languages that have been used to specify any of the emulators required by any of the documents in its collection. Since these specification languages are not expected to change very often, this should not be a significant burden.

5) Whenever a new emulator specification language is developed, ensure that a new emulator specification interpreter is developed for it, to run on whatever is the current emulation virtual machine.

6) Whenever a new emulation virtual machine is developed for emulator specification interpretation, ensure that a program is developed for it, which emulates the previous emulation virtual machine, so that the new emulation virtual machine will be able to run all previous emulators.

7) Ensure that the current emulation virtual machine always runs on some currently available computing platform. This may require porting or funding the porting of this virtual machine to new computing platforms.

Note that these steps do not actually provide access to the preserved digital documents. It is a curious feature of the emulation-based approach that preservation can be provided independently of access: in particular,

documents that are never accessed can be preserved with a minimum of effort.

8.3.1 Contrast with Migration

The ongoing preservation process is the most labor-intensive aspect of migration. Not only must bit streams be copied to new storage media, but documents of each distinct format must be converted into new formats on a frequent basis whose timing is independent of the timing of other formats or of the need to copy bit streams to new storage media. This conversion process must be applied to every document in the repository and may require specialized processing for particular documents. In order to perform conversion, the agency must maintain running versions of software to access each old format that is still in use, as well as each new format into which these old formats are to be converted. It must also develop specialized conversion programs for each such conversion and must maintain sufficient processing power to apply such conversions to every document whenever it needs to be converted into a new format.

8.4 Providing access to preserved digital documents

In order to access a preserved document, a preservation agency must either:

1) Access the digital-original by running its original software under emulation, using the current emulation virtual machine on some current computing platform.

or

2) Generate a vernacular use-copy of the document from its digital-original, by rendering the digital-original using emulation and then extracting a use-copy from the emulation environment running on the current emulation virtual machine. After extracting this use-copy (once for a given epoch of use) subsequent access to the use-copy requires running current (vernacular) software.[49]

---

[49] Such use-copies can be saved or discarded (to be generated again if needed), at the discretion of the agency.

8.4.1 Contrast with Migration

Standards/migration based preservation is incapable of providing access to a digital-original, since this becomes unusable (and may be discarded) the first time a digital document is converted – which may even be prior to or during ingestion into the repository. The approach provides vernacular use-copy access only, since all documents are converted into successive vernacular versions over time. Moreover, this approach treats the use-copy as if it were a preservation copy: that is, it attempts to convince users that they are seeing the original digital document. Further, it is incapable of showing users how the use-copy version they are seeing differs from the original, since the original is unusable. Providing access to the use-copy requires running current (vernacular) software.

9. COST COMPARISON

Although some proponents of standards/migration based approaches have claimed that emulation would be more expensive than migration, the truth appears to be quite the opposite. The main cost advantage of emulation is that virtually no processing whatsoever need be done on individual digital documents to continue to preserve them, once they have been accepted into a repository. In contrast, migration and standards based approaches require expensive and repetitive conversion of every individual document into successive new formats as old ones become obsolete. Some up-front packaging must be performed for each document that is to be preserved by means of emulation, but in most cases this will consist simply of creating appropriate metadata and encapsulating the document, both of which must be done for most other schemes as well.

In some cases, as discussed above, documents to be preserved via emulation may require the creation of installation memory images, representing a specific software and hardware configuration required to run the document's original software. This cost has no direct analogue in migration, but only because such documents are incapable of being preserved by migration; the best that a standards/migration approach can do with such documents is require that they be converted into some other, acceptable 'ingestion' format, which is likely to require more processing and expense than the configuration packaging required under emulation. And while a small percentage of documents may require such configuration packaging for preservation under emulation, most will require virtually none, since they will utilize standard configurations. The average initial per-document cost for emulation should therefore be very low.

While the initial per-document cost of preserving a document under emulation should on average be very low and the ongoing cost is virtually zero, there are additional costs to be considered. Each application (corresponding to a given logical format) that is used for any preserved digital documents incurs a small cost under emulation, since that application's bit stream must be saved at least once, and its name must be registered for reference from the metadata of those documents that require it. Similar costs are incurred by migration for each application that it utilizes to render its

converted digital documents. In fact, the per-application cost incurred by migration is likely to be greater than that for emulation, since each individual application must be ported to each new computer platform on which it is to run under migration (though no single standard is likely to last for more than a relatively small number of ports); however, since a standards/migration based approach relies on a relatively small number of standard applications, this cost would be multiplied by a smaller factor than is the case for emulation, one of whose strengths is that it can support many different applications.

Emulation also incurs a cost for each original platform on which some digital document of interest ran: this is the cost of producing an emulator specification for each such platform. Similarly, the emulation virtual machine must be ported to each new platform on which older emulators are to run. These may be non-trivial costs, but since they occur only once per platform, they should have far less effect on overall cost that the high per-document cost of migration.

Finally, it is interesting to contrast both emulation and standards/migration based preservation with a minimalist technique that has been informally referred to as 'digital archaeology'. This approach advocates saving the bit streams of digital documents without any software or hardware specifications at all. These bit streams are copied as necessary but are not interpreted or processed in any way until it is desired to access a document. At that point, retrospective analysis and processing are performed on the document to try to interpret it. (It is tempting to refer to this as 'just-in-time' preservation, except that it may very well *not* occur in time but rather be too late to preserve a given document.) This approach has an even lower up-front cost than emulation and just as low an ongoing per-document cost (i.e., essentially zero, except for copying bit streams). It also avoids any per-application or per-platform cost (though it will have a much better chance of being useful if application bit streams and platform descriptions are saved as well, to be analyzed when necessary in the future). In any case, this is clearly as minimal an approach as possible while retaining any chance at all of being able to read old documents. The trouble of course is that it is unlikely than any given document will actually be usable in the future using this technique. If the number of documents that are likely to be accessed is vanishingly small, then perhaps this approach makes sense; but then it may be no more effective

(while being definitely more expensive) than simply throwing digital documents away rather than attempting to preserve them at all.

10. REQUIRED RESEARCH

The discussion above gives some indication of the state-of-the-art of emulation and discusses some of what is needed to develop it into a serious preservation approach. In order to prove the feasibility of this approach, further research is required in three areas: (1) techniques must be developed for specifying emulators that will run on unknown, future computing platforms and which will validly (and verifiably) reproduce the behavior of original platforms; (2) techniques must be developed for keeping explanations human-readable in the future; and (3) techniques must be developed for encapsulating documents, software, emulator specifications, and associated metadata to ensure their mutual cohesion and prevent their corruption; (4) techniques must be developed to help future users use obsolete software and emulated hardware. The following discusses each of these in turn.

10.1 Emulator specification formalism

Although the 'layered emulation' approach can be implemented immediately, an emulator specification formalism should ideally be developed that captures all relevant attributes of a hardware platform, including interaction modes, speed (of execution, display, access, and so forth), display attributes (pixel size and shape, color, dimensionality, and so forth), time and calendar representations, device and peripheral characteristics, distribution and networking features, multiuser aspects, version and configuration information, and other attributes. The formalism must be extensible so that future attributes can be added when needed (for example, for compliance with future Y10K standards). The set of attributes needed to ensure that a future emulation precisely reproduces an obsolete platform in all possible aspects is unbounded, but the scheme assumes that a certain degree of variance in the behavior of emulators will be acceptable. This variance corresponds to that in the original program's behavior when executed on different contemporary systems and configurations, using different monitors, keyboards, disk drives, and other peripheral devices.

This research should include the development of validation tests to allow verifying that an emulator correctly reproduces all relevant aspects of the platform it is emulating. This should rely primarily on 'validation suites' that

can be performed on the emulator itself, independent of what obsolete software is being run on it or what digital documents are being rendered by that software. However, software-specific tests should also be investigated, as should tests that utilize 'benchmark' examples of various digital document types, to verify that all of their relevant attributes are preserved under emulation.

10.2 Human-readable annotations and explanations

Ideally, an emulation scheme should be self-describing: that is, a suitable program running on a future computer, when asked to access a document saved using this scheme, should automatically interpret the saved explanations to find out how to open the encapsulation, generate the required emulator (or find one that has already been generated for this type of computer), and run the document's saved software under this emulator to access the document itself. Alternatively, a user could interpret the saved explanations to perform the same steps. In either case, the key to successfully accessing a document saved using the emulation approach lies in the saved explanations that accompany the document, including explanations of how to use the encapsulation itself, user documentation, version and configuration information for all the software that is to be run under emulation (and for the emulated hardware), and the emulator specification itself. Whether or not these saved explanations can be automatically interpreted by future computer programs, they must remain readable by future humans, to ensure that saved documents are not lost.

Even if the encoding of this explanatory material is standardized, whatever standard is chosen will eventually become obsolete, so the emulation strategy must allow such explanations to be converted when necessary. In order to guarantee that this conversion is performed without loss, we must develop subset-convertible encodings, which we define as having the property that if some encoding Y is subset-convertible into another encoding Z, then anything expressed in Y can be converted into a subset Zy of Z, and anything in the resulting subset Zy can be converted back into Y without loss. This allows Z to be a superset of Y (not limited to Y's expressivity) while ensuring that anything that is expressed in Y can be converted into Z and back into Y again without loss. A sequence of such encodings, evolving as necessary over

time, will solve the readability problem for explanations: each encoding in this sequence serves as an explanatory standard during a given epoch. Although it is logically sufficient, having asserted that an encoding Y is subset-convertible into encoding Z, to convert a document from Y to Z and discard the original Y-form of the document, this may not convince skeptical future users. It is therefore also important to develop the concept of a subset-converter (consisting in each case of a table or a process, depending on the complexity of the conversion) that shows how to convert Y into Z and back again. If this converter is saved, along with definitions of encodings Y and Z and the Y and Z forms of all converted information, then any future user can verify that Y is indeed subset-convertible into Z, that the information was correctly converted from Y to Z, and that nothing was lost in this conversion (by verifying that the reverse conversion reproduces the original, saved Y-form of the information). In order for all of this saved information (encodings, converters, history of conversions that have been performed, and so forth) to remain readable in the future, it must be stored using this same scheme, that is, it must be encoded in a current explanatory standard, to be subset-converted as needed in the future.

10.3 Encapsulation techniques

How do we encapsulate all of the required items that comprise a preserved digital document so that they do not become separated or corrupted and so that they can be handled as a single unit for purposes of data management, copying to new media, and the like? While encapsulation is one of the core concepts of computer science, the term carries a misleading connotation of safety and permanence in the current context. An encapsulation is, after all, nothing more than a logical grouping of items. For example, whether these are stored contiguously depends on the details of the storage medium in use at any given time. The logical shell implied by the term encapsulation has no physical reality (unless it is implemented as a hardened physical storage device). And while it is easy to mark certain bit streams as inviolate, it may be impossible to prevent them from being corrupted in the face of arbitrary digital manipulation, copying, and transformation.
Techniques must therefore be developed for protecting encapsulated documents and detecting and reporting (or correcting) any violations of their encapsulation. In addition, criteria must be defined for the explanatory

information that must be visible outside an encapsulation to allow the encapsulation to be interpreted properly.

Many encapsulated digital documents from a given epoch will logically contain common items, including emulator specifications for common hardware platforms, common operating system and application code files, software and hardware documentation, and specifications of common annotation standards and their converters. Physically copying all of these common elements into each encapsulation would be highly redundant and wasteful of storage. If trustworthy repositories for such items can be established (by libraries, archives, government agencies, commercial consortia, or other organizations), then each encapsulation could simply contain a pointer to the required item (or its name and identifying information, along with a list of alternative places where it might be found). Different alternatives for storing common items may appeal to different institutions in different situations, so a range of such alternatives should be identified and analyzed.

There is also the question of what should go inside an encapsulation versus what should be presented at its surface to allow it to be manipulated effectively and efficiently. In principle, the surface of an encapsulation should present indexing and cataloging information to aid in storing and finding the encapsulated document, a description of the form and content of the encapsulated document and its associated items to allow the encapsulation to be opened, contextual and historical information to help a potential user (or document manager) evaluate the relevance and validity of the document, and management information to help track usage and facilitate retention and other management decisions. All of this information should be readable without opening the encapsulation, since none of it actually requires reading the encapsulated document itself.

It is logically necessary only that the tip of this information protrude through the encapsulation: there must be some explanatory annotation on the surface that tells a reader how to open at least enough of the encapsulation to access further explanatory information inside the encapsulation. Even this surface annotation will generally not be immediately human-readable, if the encapsulation is stored digitally. If it happens to be stored on a physical medium that is easily accessible by humans (such as a disk), then this surface annotation might be rendered as a human-readable label on the physical exterior of the storage unit, but this may not be feasible. For example, if a

large number of encapsulations are stored on a single unit, it may be impossible to squeeze all of their surface annotations onto the label of the unit. So in general, even this surface annotation will be on a purely logical 'surface' that has no physical correlate. The reader of this surface annotation will therefore be a program rather than a human, though it may quickly deliver what it reads to a human. It must therefore be decided how such surface annotations should be encoded, for example, whether the explanatory standards described above are sufficient for this purpose or whether a hierarchy of such standards – corresponding to different levels of immediate human-readability – should be developed.

10.4 Develop 'helper' programs to aid in using old digital documents

Finally, as suggested above, it may be difficult for future users to use obsolete systems to view old digital documents. Migrated or extracted vernacular versions of user manuals or original online 'help' information might be provided, but it is worth exploring the possibility of creating expert system 'helper programs' to help users operate obsolete hardware and software.

11. CONCLUSIONS AND RECOMMENDATIONS

Preserving authentic, meaningful, and usable digital documents requires more than simply copying their bit streams to new storage media as older media become obsolete. Since such documents are essentially programs, whose logical format must generally be interpreted by appropriate application software in order to be rendered properly, preserving them entails keeping this software executable indefinitely. This would preserve a 'digital-original' that has the maximum likelihood of retaining all meaningful and relevant aspects of the document. Migration (the most commonly proposed approach to preserving digital documents) cannot achieve this. Migration's successive conversion of a digital document from one logical format into another – in addition to requiring inordinate effort applied repeatedly to every document in a corpus, regardless of how likely it is to be accessed – will inevitably corrupt the document; and since the document's original version becomes unusable, it is subsequently unavailable to correct or even detect any such corruption.

The use of emulation – in particular, using software to reproduce the behavior of obsolete computing platforms on new platforms – offers a way of running a digital document's original software in the far future. This is the only proposed approach that is both based on proven technology and at least in principle capable of preserving digital-original documents.

Although, as discussed in Section 10, additional research is required to perfect an ideal scheme for emulation, the 'layered emulation' approach described in Section 6.1.1 could be implemented immediately and would provide a viable, extensible solution that could be subsumed under a more sophisticated approach in the future, with no loss of information or wasted effort.

In order to explore the potential of emulation as a preservation technique, experimental pilot projects should be conducted to evaluate the feasibility of developing appropriate emulators suitable for use in layered emulation, as well as exploring the more advanced alternatives outlined in Section 6. Such projects should attempt to elucidate appropriate authenticity criteria for relevant disciplines, and they should develop validation tests to evaluate how well digital documents preserved by means of emulation can meet those criteria. These first steps would generate crucial evidence as to the viability of

using emulation as a sustainable, long-term solution to the preservation of digital documents.

BIBLIOGRAPHY

ACCIS. *Management of Electronic Records: Issues and Guidelines*, New York: United Nations (Advisory Committee for the Coordination of Information Systems), 1990.

ACCIS. *Strategic Issues for Electronic Records Management: Toward Open System Interconnection*, New York: United Nations (Advisory Committee for the Coordination of Information Systems), 1992.

Bearman, David. 'The Implications of *Armstrong v. Executive Office of the President* for the Archival Management of Electronic Records,' *American Archivist*, Vol. 56, 1993, pp.150-160.

Bearman, David, ed. *Hypermedia and Interactivity in Museums: Proceedings of an International Conference*, Pittsburgh, 1991.

Bearman, David. 'Documenting Documentation,' *Archivaria*, Vol. 34 (Summer), 1992.

Bikson, Tora K. 'Managing Digital Documents: Technology Challenges and Institutional Responses,' in *Proceedings of the Internal Conference of the Round Table on Archives*, Stockholm, Sweden: International Council on Archives, September 1998 (forthcoming).

Bikson, Tora K. *Strategies for Implementing Document Management Technology*, Geneva: UN Information Systems Coordination Committee (ISCC), ACC/1997/ISCC/4 - Annex II, 1997.

Bikson, Tora K. 'Organizational Trends and Electronic Media,' *American Archivist* Vol. 57(1), 1994, pp. 48-68. (Also available from RAND as Reprint RP-307.)

Bikson, Tora K. and Erik J. Frinking. *Preserving the Present: Toward Viable Electronic Records*, Den Haag: Sdu Publishers, 1993. (Parts of this book are available as RAND Reprint RP-257.)

Bikson, Tora K. and Sally Ann Law. 'Electronic Information Media and Records Management Methods: A Survey of Practices in United Nations Organizations,' *The Information Society*, Vol. 9(2), 1993, pp. 125-144. (Also available from RAND as Reprint RP-508.)

Codd, E. F. 'Relational Database: A Practical Foundation for Productivity,' *CACM*, Vol. 25, No. 2, February 1982, pp. 109-117.

Coleman, J. and Don Willis. *SGML as a Framework for Digital Preservation and Access*, Washington, DC: Commission on Preservation and Access, 1997.

Cox, Richard J. 'Re-Discovering the Archival Mission: The Recordkeeping Functional Requirements Project at the University of Pittsburgh, A Progress Report,' *Archives and Museum Informatics* 8, no. 4 (1994): 279-300.

Cox, Richard J. 'The Record in the Information Age: A Progress Report on Reflection and Research,' *Records & Retrieval Report* 12, no. 1 (January 1996): 1-16.

Day, Michael. 'Extending Metadata for Digital Preservation,' *Ariadne*, http://www.ariadne.ac.uk/issue9/metadata/ (May 1997).

Dollar, Charles M. *Archival Theory and Information Technologies: The Impact of Information Technologies on Archival Principles and Practices, Information and Documentation*, Series #1, Oddo Bucci, ed., Macerata, Italy: University of Macerata, 1992.

'Dublin Core Metadata,' http://purl.org/metadata/dublin_core

Erlandsson, Alf. 'The Principle of Provenance and the Concept of Records Creator and Record: Legal Development,' in *The Principle of Provenance*, Stockholm: Swedish National Archives, 1994, pp. 33-49.

Erlandsson, Alf. *Electronic Records Management: A Literature Review*, International Council on Archives' (ICA) Study, 1996, http://www.archives.ca/ica, ISBN 0-9682361-2-X

Fonseca, F., P. Polles and M. Almeida. *Analysis of Electronic Files of Bank Reports*, Washington, D.C.: The World Bank, Business Unit Document Services, December 1996.

Fox, E. A. and G. Marchionini (guest eds.). 'Toward a World Wide Digital Library,' *CACM*, Vol. 41, No. 4, Jan. 1998, pp. 28-98.

Hedstrom, Margaret. 'Understanding Electronic Incunabula: A Framework for Research on Electronic Records,' *The American Archivist*, Vol. 54, No. 3 (Summer 1991), pp. 334-54.

Hedstrom, Margaret. 'Electronic Records Program Strategies: An Assessment,' *Electronic Records Management Program Strategies*, 1-9. ed. Margaret Hedstrom. Archives and Museum Informatics Technical Report No. 18, Pittsburgh, PA: Archives and Museum Informatics, 1993.

Horsman, Peter. 'Taming the Elephant: An Orthodox Approach to the Principle of Provenance,' in *The Principle of Provenance*, Stockholm: Swedish National Archives, 1994, pp. 51-63.

IEEE, *Proceedings of the Second IEEE Metadata Conference*, NOAA Complex, Silver Spring, Maryland, September 16-17, 1997.

Indiana University Electronic Records Project. 'A Methodology for Evaluating Recordkeeping Systems,' draft of March 27, 1997.

Kenney, Anne R. 'Digital to Microfilm Conversion: A Demonstration Project,' http://www.library.cornell.edu/preservation/com/comfin.html

Lancaster, F. W. *Vocabulary Control for Information Retrieval*, Arlington VA: Information Resources Press, 1986.

Lesk, Michael. 'Preserving Digital Objects: Recurrent Needs and Challenges,' http://community.bellcore.com/lesk/auspres/aus.html

Lesk, Michael. *Preservation of New Technology: A Report of the Technology Assessment Advisory Committee to the Commission on Preservation and Access*, Washington, 1992.

Manes S. 'Time and Technology Threaten Digital Archives...' *New York Times*, 4/7/98, Science Times section, p. F-4.

'Metadata Attribute Recommendations,' http://www.lis.pitt.edu/~nhprc/xhibit4.html

Michelson, Avra, and Jeff Rothenberg. 'Scholarly Communication and Information Technology: Exploring the Impact of Changes in the Research Process on Archives,' *American Archivist*, Vol. 55, No. 2 (Spring), 1992.

Morelli, Jeffrey D. 'Defining Electronic Records: Problems of Terminology,' in *History and Electronic Artefacts*, Edward Higgs, ed., Oxford: Clarendon Press, 1998, pp. 169-183.

Morris, R. J. 'Electronic Documents and the History of the Late Twentieth Century: Black Holes or Warehouses,' in *History and Electronic Artefacts*, Edward Higgs, ed., Oxford: Clarendon Press, 1998, pp. 31-48.

NARA. Management, *Preservation and Access For Electronic Records with Enduring Value*, July 1, 1991.

National Research Council. *Preserving Scientific Data on our Physical Universe: A New Strategy for Archiving the Nation's Scientific Information Resources*, Washington: National Academy Press, 1995.

Popkin, J. and A. Cushman. *Integrated Document Management – Controlling a Rising Tide*, Stamford CT: Gartner Group, 1993.

Roberts, D. 'Defining Electronic Records, Documents and Data,' *Archives and Manuscripts*, Vol 22., No. 2, 1994, pp. 14-26.

Rothenberg, Jeff, and Tora Bikson. *Carrying Authentic, Understandable and Usable Digital Records Through Time*, RAND-Europe, http://www.archief.nl/digiduur/final-report.4.pdf (1999).

Rothenberg, Jeff. *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation*, A Report to the Council on Library & Information Resources (CLIR), http://www.clir.org/pubs/reports/rothenberg/pub77.pdf (1999).

Rothenberg, Jeff. 'Metadata to Support Data Quality and Longevity,' First IEEE Metadata Conference, April 16-18, 1996, Silver Spring, MD, published only at http://computer.org/conferen/meta96/rothenberg_paper/ieee.data-quality.html (1996).

Rothenberg, Jeff. 'Ensuring the Longevity of Digital Documents,' *Scientific American*, Vol. 272, No. 1, January 1995, pp. 24-29.

Sanders, Terry. *Into the Future*, video/film (60 minute version shown on PBS; 30 minute version also available), American Film Foundation, 1333 Ocean Ave., Santa Monica, CA 90406 (310/459-2116), 1997.

Schurer, Kevin. 'The Implications of Information Technology for the Future Study of History,' in *History and Electronic Artefacts*, Edward Higgs, ed., Oxford: Clarendon Press, 1998, pp. 155-168.

Smith, Abby. 'Preservation in the Future Tense,' CLIR Issues, No. 3 (May/June), 1998, published by the Council on Library & Information Resources (CLIR), Washington, D.C.

Swade, Doron. 'Preserving Software in an Object-Centred Culture,' in *History and Electronic Artefacts*, Edward Higgs, ed., Oxford: Clarendon Press, 1998, pp. 195-206.

Task Force on Archiving Digital Data. *Preserving Digital Information*, commissioned by the Commission on Preservation and Access, and Research Libraries Group. May 1, 1996.

Thibodeau, Kenneth. 'To Be Or Not to Be: Archives for Electronic Records,' in *Archival Management of Electronic Records, Archives and Museum Informatics Technical Report* No. 13, ISSN 1042-1459, David Bearman, ed., 1991.

'Time & Bits: Managing Digital Continuity,' http://www.ahip.getty.edu/timeandbits/intro.html

United States Congress, House Committee on Government Operations. *Taking a Byte out of History: The Archival Presentation of Federal Computer Records*, House Report no. 101-987, Washington, 1990.

United States District Court for the District of Columbia: Opinion of Charles R. Richey, United States District Judge, January 6, 1993.

Van Bogart, John W. C. 'Long-Term Preservation of Digital Materials,' presented at the National Preservation Office (NPO) Conference on *Preservation and Digitisation: Principles, Practice and Policies*, held September 3-5, 1996 at the University of York, England.