Digital Preservation Testbed White Paper

Emulation: Context and Current Status

The Digital Preservation Testbed was founded by the National Archives and the Ministry of the Interior and Kingdom Relations. It was established in October 2000 to research different methods of digital preservation over the long term. The Digital Preservation Testbed is part of the ICTU, a government initiative which houses different research projects concerned with varying aspects of e-government.

ICTU
Nieuwe Duinweg 24-26
2587 AD Den Haag

Tel. 070 888 77 77
Fax: 070 888 78 88

E-mail testbed@nationaalarchief.nl
www.digitaleduurzaamheid.nl

Digital Preservation Testbed White Paper *Emulation: Context and Current Status.*

Den Haag, June 2003.

Foreword

This is the third and last white paper of the Digital Preservation Testbed dealing with preservation strategies: *"Emulation: Context and Current Status"*
The white papers concerning 'migration' and 'XML' have preceded. This white paper presents the current status of emulation and gives an overview of the questions that still need further investigation.

This white paper could not have been realized without Jeff Rothenberg from RAND corporation. I would like to thank him and the rest of the Testbed team for the work they have carried out writing this white paper.

I hope this white paper contributes to the (inter)national research world concerning digital preservation and to further research and practical possibilities of emulation in particular.

Jacqueline Slats
Program Manager
Digital Preservation Testbed

Contents

1 *Introduction*[1]

Digital Records are fragile. The debate as to the best means of preserving digital records over the long term has been underway for many years and will no doubt continue for years to come[2]. Various theoretical solutions have been proposed, and research is currently underway around the world to identify ways in which digital records can be authentically maintained whilst remaining accessible and usable over the long term. This paper focuses on the use of emulation in digital preservation. We place emulation in context with contemporary thinking and practice about digital preservation, identify the issues involved, and provide a summary of current knowledge and research into emulation.

1.1 Digital Preservation Defined

Digital Preservation is concerned with ensuring that records which are created electronically using today's computer systems and applications, will remain available, usable, and authentic in ten to one hundred years time, when the applications and systems which were used to create and interpret the record will, more likely than not, no longer be available. Digital preservation consists of preserving more than just the record's bit stream. We must also be able to *interpret* the bit stream in order for the *record* to survive. Without interpretation, the bit stream is nothing more than a meaningless series of 0's and 1's. During preservation, questions of record context, content, structure, appearance, and behaviour must also be taken into account. Appearance and behaviour are aspects that are peculiar to digital records. These may therefore require the most attention to authentically preserve the record over the long term.

There is a wide range of digital formats available and, to make matters more complicated, different digital objects have different preservation requirements. These can depend on the reason the record is being preserved, how long it needs to be preserved, the context and history of the record, and its original format[3]. Digital Preservation does not mean the same thing for each digital object. Whilst it is often considered that digital preservation means preserving the object so that it is identical to its original format, this is not always required. It is not always necessary to preserve every aspect of a digital record, and thus research is underway to define the essential aspects of records and their authenticity requirements. In all cases, however, the record must be preserved so that it retains its integrity and is authentic and usable. This presents interesting challenges.

---

[1] The text of this introduction has been taken nearly unchanged from the earlier published report "Migration: Context and Current Status" / Digital PreservationTestbed, Den Haag, December 2001.

[2] The phrase long term can mean fifty years or more, as indicated by Bennett in *A Framework of Data Types and Formats, and Issues Affecting the Long Term Preservation Of Digital Material* (1997). This appears to us to be a reasonable amount of time for which to prepare a long term preservation strategy. Technological changes after fifty years may well exceed expectations and limit the validity of a well-designed strategy. However, we must bear in mind that Dutch National Archives Regulations, specifically article 11 (1995), speak of a time scale of at least one hundred years.

[3] Dutch regulations (article 8) stipulates that what needs to be preserved depends on the requirements of the working process to which the record belongs.

1.2 The Task of Achieving Digital Preservation

There is a difference between paper and digital records. Any paper record can be perceived through the five human senses; no digital record can be perceived without going through computer hardware and software. For this reason, the speed of technological obsolescence makes digital preservation an important issue for everyone.

Digital records are software dependant. They rely upon the software that was originally intended to interpret (or display) them. When that software becomes obsolete, perhaps within the space of a few years[4], the problem arises of how to read that record without its original software application. It is unlikely that different versions of the application will read the file in the same way, and this may well result in a change in the interpreted record (the visible or available view of the file) that affects its archival integrity. Some data may be lost altogether; in other areas, data may be gained. There may be no way to compare a new version with the original, so changes may go unnoticed. Any changes to the record may affect its authenticity and integrity, which in turn may affect its archival and legal status. Depending on the nature of the record and its use, this can cause problems, not least that of losing or misrepresenting history.

Even a simple office computer system uses several different software applications. For each application, there may be several software manufacturers offering their own products. The rate at which new versions of software are released, with extended and (not necessarily backward compatible) new features, adds to the problem. Take Microsoft's Word® application as an example. The past six years have seen 4 versions released: Word 95; Word 97; Word 2000; and Word 2002. Two other versions of Word have also been produced for non-Windows Operating Systems – Word 98 Special Edition for Apple and IMac, and Word 2001 for the Mac. There are also different releases of Word within these versions. These releases have fewer differences among them, but each has the potential to affect a record's integrity or authenticity.

There are already many examples of how quickly digital records and data can become inaccessible. The specific details concerning who sent the first e-mail communication in the 1960's are no longer available. Some records from the old East German Republic have been lost forever, through technological obsolescence. A recent communiqué on the Joint Information Systems Committee (JISC) listserv revealed articles describing NASA's loss of data from the Viking probes sent to Mars in the mid 1970's[5].

There are several strategies for digital preservation. The following section provides a brief analysis of seven preservation approaches.

---

[4] Gail Hodge, in *Best Practices for Digital Archiving* (1999), states that "new releases of databases, spreadsheets, and word processors can be expected at least every two to three years, with patches and minor updates released more often". The Public Record Office at Kew state that it would be unusual if migration occurred more frequently than every three years, in their *Guidelines on the Management, Appraisal and Preservation of Electronic Record* (1999)

[5] JISC listserv, Friday 3rd August 2001, A *nice case study for digital preservation.*

1.3 Different Approaches

The main preservation strategies are: *technology preservation; printing to paper; migration; encapsulation; virtual machine software; XML; storage in standard formats;* and *emulation*. These strategies have different technical requirements and costs. They also have different preservation metadata requirements.

1. *Technology Preservation*. One of the first options to be used was to preserve the technology required to access original records for as long as those records are required. However, this is costly and technologically complex (although in practice, some large corporations continue to employ this approach). Support for the software and hardware eventually ceases and the parts required to maintain the hardware become more and more scarce as manufacturers discontinue obsolete components. The number of machines available that are capable of reading old files continues to decrease, for computers do not last forever. The skills required to operate the hardware and software also become rare and eventually disappear.

2. *Printing to Paper.* This is another of the early approaches which is also still in practice. However, printing all records to paper is not a viable preservation method for the majority of records. Printing to paper loses functional or behavioural traits that the records had in their digital form. Certain information may also be lost. Embedded formulas in a spreadsheet, for example, will not print to paper. Databases were simply not designed to be printed out, and any printed version is only a selective view of the database, and not a preserved format. Legal rulings have worked both for and against printing to paper[6]. As the NLA notes, 'flat data', such as text and some still images, can be printed to paper without loss of data but with some possible loss of functionality[7]. Printing to paper is often employed as an interim approach to preservation whilst a digital solution is sought.

3. *Encapsulation*. In contrast to the migration approach, the encapsulation approach retains the record in its original form, but encapsulates it with a set of instructions on how the original should be interpreted. This would need to be a detailed formal description of the file format and what the information means. This encapsulating layer could be expressed using XML, for example. If the original software used to interpret the data file is complex, then the description must also be complex and care would need to be taken to ensure that it was sufficiently complete. An extension to this idea is to create this description with an executable program: that is the subject of the section "Virtual Machine Approach".

4. *Virtual Machine Software*. A variant of the emulation approach has been proposed by Raymond Lorie of IBM[8]. This addresses the problem of interpreting data files in the future by writing a program to carry out this interpretation in the machine language of a "Universal Virtual Computer" (UVC). This program would be written at the time the record was archived and would be preserved together with the record. This program runs on what Lorie calls a UVC Interpreter, i.e. a virtual machine. In order to interpret the record on a future computer, a UVC Interpreter would be required and this could be produced from the specifications

[6] The saga concerning NARA's GRS20 went on for many years, with the Judge initially ruling against GRS20, stating that an email document is not the same as a paper document, and 'that hard copy printouts of an email may omit important parts of the electronic version'. However, this ruling was later overturned by the Court of Appeal in favour of the Archivist.
[7] National Library of Australia *Draft Research Agenda* 1998, p2.
[8] Raymond A. Lorie, *Long Term Preservation of Digital Information (2000)*

of the UVC. This approach is similar in principle to that used by the Java ™ platform to achieve present day interoperability of Java programs. To make this efficient and achievable, the key features of the proposed UVC language are that it should be simple enough that it is relatively straightforward to produce the future virtual machines, and it should be general enough that it can be widely used for archiving purposes, so that it is cost-effective to produce the future virtual machines. With this approach, the data can be stored in any format and the knowledge required to decode it is encapsulated in the UVC program.
The approach can be extended to apply to the archiving of a program: this is more like the full emulation approach. It allows the emulator to be written in the UVC language at the time of archiving, without requiring any knowledge of the future target machine.

5. *XML* XML stands for eXtensible Markup Language. It is a text-based markup language for describing the structure and meaning of data. Because it is text-based, it is human readable, but it is designed primarily to be easy to process using computers. It is an open standard defined by the World Wide Web Consortium and is not tied to any particular type hardware or software. Conversion of records to XML format can be seen as a particular type of migration. However, it is often regarded as a very promising present day data format for archiving and interoperability and so deserves to be considered as an approach in its own right.

   There is a variety of ways in which XML could de used in electronic archiving. XML could be particularly useful in storing metadata and linking that metadata to the data files making up a record. The XML Stylesheet Language (XSL) is a part of the XML standard and is a way of defining the appearance of an XML document. The combination of XML and XSL is a promising method for defining both the content and appearance of document-based records. This is one aspect of the "storage in standard formats" approach, included with migration. The Testbed project has dedicated another white paper to this subject: XML and digital preservation (September 2002).

6. *Migration (including Storage in standard formats).* As has become apparent in recent reports, this is the best known and most widely applied preservation strategy[9]. It is also the most criticised method. Within the scope of the Testbed, migration is defined as the transfer of files from one hardware configuration or software application to another configuration or application. A simple example of this is the migration of a file from Microsoft Word 6 to Word 7. A more complex example is the migration of a file from an Apple Macintosh to Microsoft Windows. A frequently heard objection to migration is the fact that the results are often unpredictable, mostly because of a lack of or because the process has not been fully tested. When a new software version comes on to the market, many people carry out a straightforward update of their documents. Not infrequently, this leads to a loss of information, whether this relates to the contents, structure, appearance or context of the file. The new software does not always read the file in the same way as the original software, with the consequence that the contents and functionality can be lost. The results of migration are difficult to predict, unless a substantial amount of work is first done regarding the specifications of the source

[9] The InterPARES report, *Preservation Strategies for Electronic Records, Round 1 (2000-2001) Where We Are Now: Obliquity and Squint?* (2001), features the results of a 2000-2001 survey of recordkeeping institutions, in which 4 out of 13 projects identified migration as their preservation strategy. This was the most prevalent approach. See also Margaret Hedstrom, Digital Preservation: Problems and Prospects (2001); also Jeff Rothenberg and Tora Bikson, *Digital Preservation: Carrying Authentic, Understandable and Usable Records through Time* (1999).

and target formats. Migration can influence the authenticity of a document. Each document that is preserved must be preserved 'authentically', because otherwise the meaning and validity of the archival record cannot be guaranteed. This has both legal and archivistic implications. The Testbed project has dedicated another white paper to this subject: Migration: Context and Current Status (December 2001).

7. *Emulation.* The theory behind Emulation is that the only way to ensure the authenticity and integrity of the record over the long term is to continue to provide access to it in its original environment, i.e., its original operating system and software application Emulation is the focus of this paper and is now discussed in more detail.

2 *The dimensions of preserving digital archival records*

Preservation involves more than simply preserving the binary digits (bits) that represent digital records: the authenticity and meaning of the records must also be preserved. That is, we must:

> Enable reliable, authentic, meaningful and accessible records to be carried forward through time within and beyond organizational boundaries for as long as they are needed for the multiple purposes they serve[10].

Beyond the technical aspects of this problem, there are administrative, procedural, organizational, and policy issues surrounding the management of digital records. Numerous features of digital records distinguish them from traditional paper records in ways that have significant implications for how they are generated, captured, transmitted, stored, maintained, accessed, and managed. Paramount among these differences is the greatly reduced lifetime of digital records without some form of active preservation: this mandates new approaches to saving digital records to avoid their loss. Non-technical issues include questions of jurisdiction, responsibility for various phases of the existence of digital records, funding, and policies requiring government agencies to adhere to standard techniques and practices to prevent loss of digital information.

However, it is difficult to address these non-technical issues meaningfully in the absence of a defined technical solution to the problem of preserving digital artifacts. Furthermore, any technical solution must be able to cope with issues of corruption of information, privacy, authentication, validation, and preserving intellectual property rights[11]. Finally, any technical solution must be feasible in terms of the societal and institutional responsibilities and costs required to implement it.

In the government archiving context, any solution to the technical preservation of digital artifacts is merely a means to the end of preserving authentic digital archival records. Authenticity is difficult to define precisely and may be different for different kinds of records in different business process contexts, leading to different preservation criteria.

[10] See [13].
[11] This is especially complex for records that are "born digital" and therefore have no single original instance, since traditional notions of copies--on which much of copyright law rests--may be inapplicable to such records.

3 *Interpreting and rendering preserved digital records*

By far the most challenging technical aspect of preserving digital records, documents or other artifacts is that they require interpretation in order to be made perceptible to humans. Just as hieroglyphics remained unintelligible for thirteen centuries prior to the discovery of the Rosetta Stone, the streams of bits that represent digital records will be unintelligible to future generations unless we preserve ways of interpreting them correctly.[12] In the digital realm, this interpretation results in what is often called "rendering" and denotes the process of turning a stream of bits into something that humans can see, hear or otherwise experience. If a bitstream cannot be appropriately rendered, it may as well be discarded as meaningless.

The intended meaning of a bitstream is implied by its "logical format" which is a set of conventions that explain how its bits are to be interpreted[13]. For example, a bitstream in 8-bit ASCII format consists of groups of 8-bit sequences (called "bytes") each of whose first bit is zero and whose remaining 7 bits represent one of the 128 typographic symbols and "control characters" defined by the ASCII standard. On the other hand, a simple bitonal, bitmapped graphic format to represent small icon images might
consist of a 256-bit stream intended to be interpreted as describing a square graphic image made up of 16 rows of 16 cells, where each cell is either black or white depending on whether the corresponding bit is 1 or 0. Note that two bitstreams representing these two quite different formats may look exactly the same. There may be no way to tell from looking at a given bitstream which format it embodies; in fact, the same stream of 256 bits (assuming that each 8th bit were zero) might be interpreted either as a string of 32 ASCII characters or as a single icon image. As this example illustrates, the logical format of a bitstream cannot generally be inferred from the bitstream itself [14]. Generally speaking, in order to know how to interpret and render a bitstream correctly, it is necessary to understand every detail of its format.

Given a bitstream and a suitable explanation of its format, a human can read the bitstream and interpret it by hand in order to render it. For example, a user could read sequences of 8-bit bytes and interpret each one using a table of ASCII codes; the user could then write the resulting characters on paper for other users to read. However, performing this kind of manual interpretation of any but the shortest bitstreams in the simplest of logical formats is so laborious, time consuming and error-prone as to be entirely unrealistic.

---

[12] . It is logically provable that an arbitrary stream of symbols (and there are few things more arbitrary than a sequence of zeros and ones) cannot be made self-explanatory without some additional, external information that explains what it represents and how it represents it.

[13] . Note that complex records may consist of more than single files of bits: for example, they may link to resources such as databases, network data streams, etc. Throughout this paper, the "bitstream" of a record is taken to mean all of the bits that comprise the record, wherever they may reside.

[14] In the 8-bit ASCII case, every 8th bit is zero, which provides a clue as to the format, but the same sequence of characters could be represented in 7-bit ASCII, where the leading 0 bit of each character is omitted, in which case there would be no such clue. (Our example 256-bit stream would contain 36 7-bit ASCII characters, with 4 bits left over.)

As infeasible as manual interpretation is for simple formats like ASCII, it is completely inadequate for dynamic, active, or interactive formats. Even relatively simple graphical formats such as PostScript or PDF involve elaborate mathematical computation to draw characters of particular sizes in particular fonts on a display screen or printer page. More complex formats may involve database query, network access, animation, interaction with a user, or the execution of arbitrarily complex programs in order to generate
and render their content [15]. Formats like these may be "inherently digital" [24] meaning that digital records in these formats rely on being in digital form for some fundamental aspect of their content, meaning or behavior [16]. Manual interpretation of inherently digital records is unthinkable.

To summarize, rendering any digital format involves executing an algorithm that interprets a bitstream and produces some human-perceptible result, whether this is visual, audible, tactile, temporal or involves some combination of these and other sense modalities. Rendering algorithms may do more than simply render static content into human-perceptible form: they may actually construct the content of a digital record during execution, whether by piecing together pre-existing content elements, generating new elements ab initio, or creating dynamic behavior, such as animation. Because machines are far superior to humans at executing complex, repetitive algorithms, rendering is invariably performed by computers.

In practice then, digital records--and particularly those that are inherently digital–are usable only if their bitstreams can be interpreted and rendered by software running on suitable computers [17].

[15] Although we have emphasized rendering, complex digital formats may actually generate their contents dynamically--on the basis of database query, user interaction and arbitrary computation--prior to rendering it. For example, many web pages do not consist of existing text or images: instead, they execute scripts that generate appropriate text and imagery--or animated output--whenever they are accessed.

[16] Being inherently digital is a much stronger criterion than being "born digital" which merely means that a record happens to be in digital form --whether it needs to be or not.

[17] This software may not be a single application program running on a single computer. It may consist of many software components, utilizing library programs, operating system facilities, database or network resources, etc., executing in one or more processes on one or more computers, possibly utilizing client-server, multi-tiered or other distributed architectures across networks. For the purposes of this paper, however, we will refer to each instance of such rendering software as a single logical program, regardless of how it is implemented.

3.1 Preserve records in ways that allow interpreting them

The above argument implies that it is not enough to save digital records: we must also ensure that suitable ways of interpreting and rendering them will exist in the future. The problem of preserving a digital record can therefore be restated in terms of two questions:

1) What representation of the record should we save?

2) How can we ensure proper interpretation of that representation in the future?

As noted above, virtually all digital records are interpreted by running software. Not surprisingly, then, virtually all proposed digital preservation approaches rely on running software in the future in order to render saved digital records. Even those approaches that attempt to encode the meaning and appearance of digital records in human-readable formats like XML implicitly rely on running future programs that can render these encoded records in appropriate ways [18].

Most serious preservation approaches save some form of bitstream for each record, though few approaches utilize the record's original (or "native") bitstream [19]. Virtually all proposed approaches except "digital archaeology" and emulation rely on sooner or later transforming a record's original bitstream into some other format that is used in place of the original bitstream. This is done in the hope of simplifying future interpretation. In terms of the two questions posed above, the answer to (1) is determined by first answering (2): some format is chosen based on its likelihood of being interpretable in the future, and the original record is transformed into that format in order to be preserved.

Some approaches perform this transformation at the time a record is initially accepted for preservation, whereas others perform it later, when the record's initial file format becomes obsolete. Some approaches assume that such a transformation need only be performed once, whereas others recognize that it is likely to be necessary to perform repeated transformations into successive new file formats over time, as each digital format becomes obsolete in turn [20].

The problem with any transformation is that it introduces the danger of corruption and misinterpretation. Since each different digital format requires its own distinct interpretation, transforming one format into another should ideally be done in a way that produces identical results when the two formats are interpreted; but since no two formats have identical capabilities and characteristics, this ideal is rarely possible.

---

[18] Furthermore, all such formalisms are intended to be saved as digital files, which must be rendered, even if they are to be used manually. For example, an XML file consists of a sequence of Unicode character codes which must be rendered in order for humans to read the resulting XML. Although such files could of course be printed rather than saved digitally, doing so would defeat much of the advantage of using a computer-readable representation like XML; in particular, XML or any other encoding must be further interpreted by software to render the desired record.

[19] Many approaches save a record's original bitstream as a backup but offer no realistic way of rendering this in the future. Note that it is not always clear exactly what constitutes the native format of a record. For example, some applications provide alternative "interchange" formats for their files (such as rtf for Microsoft Word or mif for FrameMaker), which may or may not be equivalent to their normal format.

[20] This is the well-known ‘migration’ approach to digital preservation.

Even the most careful and clever transformation is likely to produce a bitstream whose interpretation renders content, structure, appearance or behavior that differs from that of the original record. Successive transformations compound this danger, making it quite unlikely that a future transformed record will be truly equivalent to the original. The implications of transformation for authenticity and record keeping are discussed further in Section 4.4 below, but first we will consider its technical motivation.

3.2 Is it necessary to transform digital records in order to preserve them?

The underlying motivation for transforming the original bitstreams of records is that it is difficult to answer question (2) above for an arbitrary digital format. That is, if records are accepted by archives in any unconstrained format, it is difficult to ensure that these formats will remain interpretable in the future, since formats become obsolete over time.

What does it mean for a format to become obsolete? Essentially, it means that it becomes difficult or impossible to run software that knows how to interpret and render that format correctly. This may occur because the format becomes unpopular or because it is superseded by some new formator new paradigm; or a format may be proprietary to some company that goes out of business. When a format becomes obsolete for whatever reason, the original software that renders that format will cease to be updated and maintained and so will eventually become unusable as its programming environment or paradigm or the hardware platform on which it runs becomes obsolete.

If enough records or other digital artifacts are encoded in a format that has become obsolete, new "viewer" programs may be written to render that format after its original rendering software becomes unusable; but this has its own problems. Writing new rendering software for a format requires a deep understanding of the details of that format. Yet the very fact that a format has become obsolete implies that the expertise required to understand it in full detail may not continue to exist for long. If such understanding is not provided by extensive documentation, it requires "reverse engineering" [21] (i.e., figuring out how the format works by studying the programs that originally rendered it), which is labor-intensive, expensive and error-prone. Furthermore, as a format's original rendering software becomes unusable, it will no longer be available to provide a baseline evaluation for the behavior of new rendering software, making it difficult to validate future viewer programs for the format.

As mentioned above, there are some preservation approaches that do not rely on transformation; that is, their answer to question (1) above is to save each record's original bitstream. One such approach (sometimes called "digital archaeology") advocates saving original bitstreams, but it avoids answering question (2) above. Instead, it, is assuming that future archivists or historians will be able to decode or decipher saved bitstreams to render them appropriately [22].

Preservation approaches based on the use of well-documented standard formats may also attempt to rely on saving the original bitstreams of records. If records are generated in these standard formats in the first place (or are in any case already in these formats when they are acquired by archives), then no initial transformation need

[21] 'Reverse Engineering'- decompilation: the attempt to trace and describe the logic in compiled programs, of which the source code has disappeared. Is anyhow a difficult task, because you can no longer make a pig out of a sausage (Pagrach 1991).

[22] The history of hieroglyphics, however, reveals that such deciphering is not generally possible without something like a Rosetta Stone.

be performed. Standard formats of this kind may include formal standards (such as TIFF or JPEG), de facto standards (such as PDF), proprietary standards (such as Microsoft Word), or evolving standards (such as XML, whose associated standards--notably, the large number of different semantic vocabularies for different domains--are likely to continue to evolve and remain open-ended for some time) [23]. Standards-based approaches assume that question (2) above will be answered by the perennial availability of software that knows how to render the standard formats in question; yet it is a virtual certainty that every standard will eventually become obsolete, at which point such approaches will require successive transformation, essentially becoming variants of migration.

The final proposed approach that avoids transformation is emulation [15, 20, 9, 11, 5]. Emulation saves the original bitstreams of records and answers question (2) above by providing a way of emulating obsolete computer platforms in the future so that the original rendering software for each obsolete format can be run indefinitely. This is discussed in detail in the following sections.

[23] A variant of the standards-based approach is the formalization of records, using mathematical or formal semantic descriptions of record content and format, possibly using XML. However, such formalisms are unlikely to be adopted as the native formats for records, since they are inherently inefficient, so this approach would again require initial transformation of records into the chosen formalism.

4 *Emulation*

The term emulation is used in computer science to denote a range of techniques all of which involve using some device or program in place of a different one to achieve the same effect as using the original. The term "simulation" is often confused with--and sometimes even used as a synonym for--emulation, but we distinguish between the two terms here by noting that a simulation describes what some other thing would do or how it would act, whereas an emulation actually does what that thing would do. For example, an airplane simulator does not actually fly. That is, simulation generally involves the use of a model to understand, predict or design the behavior of a system rather than the practical recreation of that system's capabilities[24]. In contrast, emulation is generally used to create a surrogate for the system being emulated.

A simple metaphor should clarify this distinction between simulation and emulation. An actor playing the role of a poet is simulating the behavior of the poet, whereas someone impersonating the poet is emulating that behavior. We would not expect the actor to be able to compose a poem on demand, whereas we would expect the impersonator to be able to do so. Impersonation is a good synonym for the term emulation as it is used in this paper.

In some cases, hardware devices are used to emulate similar hardware devices, often to provide standard capabilities. For example, most modems emulate a de facto standard set of modem capabilities referred to as the "Hayes modem command set" and many computer terminals (and communications programs) provide a "VT100 emulation mode" to provide the features of the Digital Equipment Corporation (DEC) VT100 terminal. In such cases, the emulating device can be used just as if it were a Hayes modem or VT100 terminal[25]. In other cases, hardware devices may be designed to emulate programs, though we usually say that such devices "implement the program in hardware" rather than emulating it. In fact, it is possible to emulate either hardware devices or programs using either hardware devices or programs, so there are four possibilities. However, for our purposes, we will focus on just one of these, i.e., writing programs that emulate computer hardware. For the remainder of this paper, we will therefore use the definition:

> Definition (a): An emulator is a program that runs on one computer (the emulator's "host" system) and makes that computer behave like a different computer (the emulator's "target" system).

Note that any program that runs on any computer can be thought of as temporarily turning that computer into a different machine. As a simple example, running a clock program on a computer temporarily turns that computer into a clock[26]. Similarly, a

[24] See [17].

[25] These examples may be unfamiliar to many readers, but anyone who has used a computer is likely to have used emulators of both of these devices without being aware of it. This very lack of awareness is a testimony to the success of emulation.

[26] Of course, most modern computers can run several programs at once, typically in different "windows" on their display screens, so the clock program really turns only a part of the computer into a clock. For illustrative purposes, therefore, imagine a program that turns the entire screen into a clock.

word processing program temporarily turns the computer on which it runs into a word processing machine. A useful way of saying this is that running the clock program creates a "virtual" clock (or "virtually creates" a clock). The term "virtual" is used in computer science to denote something which serves the function of some other thing without actually being that thing [27]. Notice that this concept of virtually creating a hardware device such as a clock (or virtually recreating it, which is the case if the device previously existed) is closely related to our definition of an emulator above. In fact, we can rephrase our definition as:

> Definition (b): An emulator is a program that runs on one computer (the emulator's "host" system) and thereby virtually recreates a different computer (the emulator's "target" sys tem).

A clock or word processing program temporarily turns the general purpose computer on which it runs into a special purpose machine, e.g., a clock or a word processor; but in the case of emulation, one computer is temporarily turned into a different computer, which is a general purpose machine whose function is to run other programs. When an emulator of a given target system is run on its host system, it virtually recreates the target system on the host system. This virtual recreation of the target sys tem can be used just as if it were the target system, so in principle, it can run any program that the target system can run: we say that such a program is "run under emulation" on the host system.

For preservation purposes, we focus on emulating older, obsolete computers on future computers. In this context, emulation would enable future computers to "impersonate" any obsolete computer, virtually recreating the obsolete computer and thereby allowing its original, obsolete software to be run in the future. This would allow the original rendering programs for obsolete digital formats to be run on future computers, under emulation.

It is important to reiterate that the approach discussed here involves using software to emulate hardware. Although this may sound difficult, emulating hardware should actually be far simpler than emulating (i.e., rewriting) the original application programs that rendered digital formats. For one thing, we need only emulate computer hardware at its logical level, not at the level o f its circuitry or its implementation. The internal details of digital hardware may be quite complex, but its external, logical behavior is relatively simple: this is the reason it can be used to run software. At this logical level, hardware is in fact generally much simpler and better specified than software [28]. For this reason, it is relatively straightforward to write programs that emulate computer hardware and to validate such programs against the logical behavior of that hardware. Writing an emulator of a given computer system (including its peripherals [29]) is not a trivial undertaking, but it is a well understood and tractable task; furthermore, only one such emulator need ever be written for any given computer.

---

[27] Virtual memory, for example, uses disk storage to achieve the effect of having more random access memory (i.e., RAM) than is actually installed in a given computer.

[28] As pointed out elsewhere [19], computer hardware must be well specified in order for it to be manufactured. In contrast, software is not manufactured: once it is working, it is simply copied, so it is rarely specified or described very well, since it does not need to be.

[29] A "peripheral" device is any piece of hardware that is used by a computer system without being integral to it. Examples include input and output devices such as displays, keyboards, mice or trackballs, printers, scanners, etc. In some cases, sound, video or other multi-media capabilities may also be provided by peripherals, as may various forms of secondary or backup storage, such as plug-in disk drives, including CD-ROM or DVD.

Emulation avoids the need to write new software in the future to render obsolete formats. This is a significant advantage, since an obsolete format must be understood in great detail in order to write such rendering programs, which may require extensive research and possible reverse engineering, if the format in question is not well documented.

The hardware emulation approach described here is the only way that has so far been proposed to run original software on future computers. This means that the behavior of that original software will be recr eated (within the limits of the emulation approach, as discussed below) without anyone needing
to understand or rewrite any of that software. None of the original rendering programs or their original operating system environments need be recreated or modified in any way: they are simply saved and run exactly as they were originally, albeit under emulation on future computers. When this original software is run under emulation in the future, it should be completely unaware that it is running on anything other than its original hardware. Running a digital record's original rendering software in this way should allow preserving and accessing the record in its original format.

4.1 History and current state of the art of hardware emulation

Emulation is a well-accepted technique that has a long history of use in computer science. Emulators are often developed by computer processor architects or manufacturers to try out a design before it is produced. However, "prospective" emulators such as these are not likely to be well suited to preserving digital artifacts, since their focus tends to be on testing processor design, predicting performance, or facilitating the development of new software for the new processor rather than running existing software efficiently. On the other hand, computer vendors have also often included "retrospective" emulators of previous systems in new computers to allow their customers to run old programs on their new machines [30]. These are explicit instances of using emulation for preservation purposes (where the artifacts being preserved were executable programs). Although emulating the behavior of a computer system's processor (also called its central processing unit, or CPU) is only part of the problem – emulating input and output devices, as well as other peripherals may be just as challenging – these cases demonstrate that emulation can be a practical and effective preservation mechanism.

Many of these examples involve emulators of one computer that run on another computer manufactured by the same vendor; but a number of commercial emulators are also available that emulate one brand of computer on another – which is more analogous to long-term preservation. The most notable of these emulate some version of the ubiquitous Intel platform on the Apple Macintosh; similarly, there are Apple Macintosh emulators that run under Microsoft Windows on Intel-based computers [31]. These products follow the approach we believe makes the most sense for preservation as well: They emulate the processor (and, where necessary, peripherals) of the target machine, but they do not emulate its system software. Instead, bona fide

[30] For example, the IBM 360 provided an emulator of the older IBM 7090 computer to enable customers to continue running old software without having to rewrite it for the new machine; some installations apparently even ran an emulator of the IBM 1401 under the 7090 emulator on the 360, in order to run 1401 applications on the 360. More recently, Apple's PowerPC Macintosh included an emulator of its previous Motorola 68000 processor, enabling old programs to run on the new machines without users even being aware that emulation was in use; significant portions of the Macintosh operating system itself remained written in 68000 code and continued to run under emulation for years.

[31] For example, Connectix, Inc. sells a product called VirtualPC, whereas Emulators, Inc. sells a p roduct called SoftMac.

(and licensed) copies of the desired system software must be obtained by the user to run on the emulated hardware platform. In addition to avoiding possible ethical (and legal) problems, this greatly increases the likelihood that a given application program will behave correctly, since it will use its normal system software rather than a recreation of that software. Although there are some limitations to these commercial emulators, they are good enough to be used by millions of users to accomplish real work on a daily basis [32].

Most current research on emulation involves the development of techniques to model the behavior of computer hardware for the purposes of understanding or improving its design or its behavior when running certain "mixes" of software. The fact that such work is typically described as simulation rather than emulation bespeaks its different orientation. For example, the work on SimOS at Stanford [8, 16] and related work on the PROTEUS system at MIT [2] fall into this category. Despite their different orientation, however, these efforts contribute valuable insights into emulation--particularly in terms of analyzing the tradeoffs between low-level emulation of a system's implementation details versus higher-level emulation of its functional behavior and by developing techniques for creating modular, configurable emulations (as discussed in Section 5.2.7).

In contrast, recent work at IBM Almaden promises to explore emulation as a preservation strategy [11, 12]. This effort focuses on the use of a virtual machine (called a "Universal Virtual Computer" or UVC), which would allow emulators to be hosted on future computers simply by hosting the UVC on those computers [33]. However, the emphasis of this work so far has been on the use of the UVC to implement an alternative preservation approach, based on saving extracted data from digital documents and allowing the data to be viewed in the future by running data retrieval programs on the UVC, so the emulation potential of the UVC has not yet been explored [34].

The CAMiLEON [6, 7, 9], and the NEDLIB projects [22, 23], have experimented with the use of emulation for preservation purposes, producing some promising results. In particular, the CAMiLEON project's success with its restoration of the 1986 BBC Domesday [35] is a strikingly successful case study in the potential use of emulation for preserving digital artifacts.

4.2 Other forms of emulation

The approach discussed here is that of using software to emulate computer hardware, on which original rendering software can then be run: for convenience in this discussion, we will refer to this as the "software-emulation-of-hardware" approach.

---

[32] Because these emulators run on computers of the same generation as those they emulate, they necessarily run more slowly than their target machines, though often not by more than a factor of 3-5, which is acceptable for most purposes (in fact, this is generally within the range of speeds exhibited by current systems of different ages and prices, all of which are considered acceptable for running most software). Execution speed should be much less of a problem when using emulation for preservation, since future machines are likely to be considerably faster than current ones.

[33] This virtual machine approach (elaborated in Section 4.7.3 below) was first suggested in [22], where it was referred to as an Emulation Virtual Machine (or EVM). The UVC can be seen as a candidate for an EVM.

[34] The UVC's data extraction approach may provide an attractive mechanism for performing vernacular extraction (or "migration on request") as discussed in Sections 4.4.3, 4.5 and 5.2.3 below.

[35] See http://www.si.umich.edu/CAMILEON/domesday/domesday.html.

Two alternative uses of emulation are sometimes discussed, both of which involve emulating software with software and which do not share most of the advantages of the software-emulation-of-hardware approach. These might be called application emulation and operating system emulation.
Application emulation consists of writing one application program to do what another application program does. In the preservation context, this is essentially the "viewer" approach, in which new programs are written in the future to render obsolete digital formats. This is different from the software-emulation-of-hardware approach: instead of writing a single emulator of a hardware platform, the viewer approach requires writing a new program (or adding a significant new piece to an existing viewer program) for every distinct digital format. Because many formats are proprietary, this entails reverse engineering each such digital format. Furthermore, this approach does not allow running a record's original rendering software.

Operating system emulation, on the other hand, is not really a meaningful preservation approach at all. Despite being discussed in numerous publications and presentations it is based on a fundamental misunderstanding of how programs run on computers. The idea is to recreate the operating system (OS) that is used by various rendering programs for various digital formats. This requires a significant amount of reverse engineering effort, but even so, the result is not a program that can run other programs, since this is not what an OS does. An OS merely provides facilities (user interfaces, file systems, interprocess communication, networking, etc.) that are used by programs when they run, and it allows invoking programs to be run (e.g., by double-clicking on their icons). An application program may use these OS facilities to access files, interact with users, or communicate with the network or with other programs, but the application program must always execute on hardware, just as the OS itself does. That is, any program must run on its expected hardware platform, regardless of whether its expected OS is also running on that platform. Computer scientists often say (perhaps confusingly) that an application program "runs on top of" (or equivalently "runs under") an OS, but all this means is that it relies on the facilities provided by that OS--it does not mean that the application "runs on the OS" in the same sense that the application runs on hardware. All programs (applications and operating systems alike) must run on hardware. Therefore, implementing an emulator of an OS does not enable us to run application programs, such as rendering programs, without also having the appropriate hardware platform--either as a physical computer or as a software-emulation-of-hardware (which, of course, must itself run on some physical computer) [36].

Finally, as noted above, it is possible to emulate hardware with other hardware. This is frequently done to create simpler, cheaper, or lower-power versions of existing computer processors, though it can just as well be done to create faster or more powerful versions of popular older machines. From a preservation perspective, this is quite similar to software-emulation-of-hardware, since it can be used to recreate a given obsolete hardware platform on which original rendering software for obsolete formats can be run in the future. Its disadvantage is merely that it is often (though not necessarily) harder and more expensive to build a hardware emulator of this sort than

[36] . One of the more popular OS emulators is a program called WINE, which recreates the Windows environment under a non-Windows OS. WINE's developers--obviously sensitive to the distinction drawn here--created its name as an acronym for the phrase "WINE Is Not an Emulator" but were still unable to prevent rampant confusion about its true nature. Actually, of course, WINE is an emulator: it is a software emulator of software. But WINE does not by itself enable rendering programs to be run, since it emulates an OS, not a hardware platform.

to write an emulator program [37]. Nevertheless, this approach might in the future become competitive with the software-emulation-of-hardware approach discussed here.

4.3 Motivations for using emulation to preserve digital records

Emulation has a number of potential advantages as a mechanism for preserving digital records. First, emulation offers the possibility of preserving executable records in executable form. Although most current digital records include little or no executable code, it seems likely that inherently digital records (which may execute code to produce dynamic, active or interactive behavior) will become increasingly common in the future. Indeed, some records will undoubtedly be executable, such as complex database query programs, decision-support software, or analysis or prediction models used to make policy [38].

In addition, since emulation does not require transforming original digital records into one or more new formats as their original and subsequent formats become obsolete over time, it avoids the cost of understanding the details of specific formats in order to know how to transform them. Even for ubiquitous, well-documented formats, transformation often requires expensive reverse engineering, whereas for less common or obscure formats, such costs may be hard to justify, leading to the abandonment of records that use such formats [39]. Emulation eliminates the need to transform each record's bit stream every time its previous format becomes obsolete. At the current rate of evolution of digital formats (especially considering the fact that new versions of a given format are not always compatible with older versions of the same format), transformations of this might otherwise have to be done every 3-10 years. The only way to make such successive transformation cost-effective is to automate it–but this incurs the risk that some records will be transformed inappropriately. Even if records are limited to specific, standard formats, it might be difficult to force all those who generate records to use those formats in standard ways. Yet any divergence from standard practice may involve unexpected usages that are misinterpreted or corrupted by automated transformation programs [40]. Emulation avoids all of these problems. Furthermore, in order to compensate for the limitations of each different preservation approach, it is often assumed that multiple approaches will have to be used in concert, each being applied to different kinds of

[37] Actually, many computer processors are "micro-coded" meaning that their true behavior is a result of programs running in the processor itself. For example, the Intel Pentium uses this technique to produce much of its behavior. Similarly, Transmeta Corporation's Transputer (among other things) is used to emulate the Intel Pentium processor in several commercial laptop computers. Such trends in computer development blur the distinction between hardware and software, though it is always the case that every program must ultimately run on some physical hardware.

[38] In fact, it has been argued [18, 21] that all digital artifacts are in a sense executable. The act of interpreting the bitstream of a digital artifact consists of performing actions that are specified by the bit patterns in that bitstream--which is precisely what is meant by executing a program. Although digital artifacts are often interpreted by application programs rather than hardware, the same is true for programs written in many programming languages, such as Java, Perl, Lisp and Prolog. No matter how many such levels of software interpretation are involved, however, every such chain of interpreters must ultimately execute on hardware.

[39] Whereas there may be a relatively small number of popular formats in current use, there are probably at least hundreds that will eventually become relevant for archival purposes over time, and there are thousands, if marginal and obscure formats are included.

[40] Examples of unexpected and inappropriate uses of formats abound. For example, it is not uncommon for users to create tables in text rather than using table-building or spreadsheet tools, or to use spreadsheets as ad hoc databases or as crude modeling and simulation environments.

digital records for different preservation purposes[41]. Whereas such diversification may be prudent as a way of avoiding too much reliance on any one approach, it carries a number of disadvantages of its own. First, it requires the parallel development and maintenance of all of the chosen preservation approaches, each of which incurs its own costs and complexities. Moreover, much of this cost may have to be replicated for each different format to which a given approach is applied, in which case this cost may be deemed unwarranted for uncommon or obscure formats, leading to the abandonment of records that use such formats. In addition, a diversified strategy of this kind would make it impossible for any single approach to benefit from the economy of scale of being applied more universally.

A diversified preservation strategy also requires analysis of each format, record type, and potentially individual record, in order to decide which preservation approach or approaches should be applied to it. Since there are already dozens or hundreds of relevant formats and record types--and since this number will only increase over time--this will require a significant categorization effort to decide which approach to apply to which records, prior to actually applying any preservation technique. By conforming to standard formats this problem will become more manageable.

Finally, a diversified strategy would make preserving and accessing digital records more complex, since each format or record type would require different ongoing preservation activities and future access mechanisms.

Therefore, although it may make sense to adopt a diversified strategy, it seems unfortunate to give up the possibility of finding a more universal approach without even trying. Such an approach would reduce or eliminate the need for categorization of records prior to preserving them, would take advantage of the economy of scale of being applicable to most or all records, would greatly simplify both ongoing preservation activities and future access mechanisms, would serve as a "catchall" solution that would preserve even those records that utilize obscure formats, and would (ideally) preserve executable records in executable form. A universal approach of this kind—if it exists--would almost by definition have to be based on preserving the original versions of digital records.

4.4 Preserving originals

One of the distinguishing features of emulation is its potential ability to preserve digital records in their original formats. We therefore examine the subject of originals here as a prelude to discussing the details of emulation. We first examine traditional original records and then draw some analogies between these and digital originals.

4.4.1 Traditional original records

Dutch archival law requires that the content, structure and form of archival records can be ascertained at all times, as far as they relate to the business process [42]. This requirement is traditionally taken to mean that it is first necessary to be able to reproduce original records without any reinterpretation (where the word "interpretation"

[41] There is considerable precedent for such diversification in the realm of traditional conservation, where each different kind of material to be conserved (books, tapestry, sculpture, architecture, etc.) may involve different chemistry and physical preservation processes. However, since all digital artifacts consist of bits, there is at least the potential for a more universal approach to conserving them.

[42] "Regeling geordende en toegankelijke staat archiefbescheiden", Feb. 2002.

in this context has its ordinary meaning, rather than the technical meaning introduced above in the discussion of rendering digital records). If a traditional record is in some obscure dialect or format or has some obscure context, it is assumed that users will have sufficient background and expertise to interpret it correctly.

This traditional reliance on originals makes perfect sense from a record keeping perspective. In whatever way one defines the authenticity of a record [43], it seems far easier to argue that an original record is authentic than to argue that a transformation of that record is authentic. In the case of traditional records, the concept of an original is well-defined. A traditional record is normally a physical object of some kind, so preserving this object preserves the record [44]. As has been argued elsewhere [26, 27] a physical original is unique in having all of its attributes: the distinction between an original and a copy or forgery is that nothing but the original will have all of these attributes. By definition, a physical original will retain all of its attributes over time, within the limits of physical preservation. So long as it retains its physical integrity, a traditional original record will automatically retain its content, appearance and structure.

It is sometimes necessary to copy an original traditional record because it may have deteriorated; but this may be a last resort from an archival perspective, since doing so loses many of the attributes of the original. If a record is purely textual, this copying process may not involve too much loss, although attributes such as the physical properties of its original paper, ink, binding etc. will generally be lost [45]. However, if a record includes images, copying it may entail more serious loss. In extreme cases, copying a deteriorating record may not result in an acceptable surrogate from a record keeping perspective: in such cases, the copy may serve to document what the record was but may not take the place of the record as a record.

On the other hand, making a surrogate copy or "vernacular rendition" of some kind may often be done to improve access or to reduce wear on the original [46]. Such surrogates should always be generated from the original record, not from other surrogates or transformations of the original. In such cases, of course, the original remains the record, while the surrogate is used merely for convenience. A vernacular rendition of a traditional record is a copy of the original in some more convenient or accessible form. For example, it may be useful to make modern vernacular renditions of records whose originals utilize old dialects or scripts that are no longer readable by non-scholarly users. Or it may be convenient to create modern paper copies (facsimiles) of records so that they can be widely disseminated or used without

[43] The term "authenticity" is used here to encompass both integrity and authentication, where "integrity" denotes that property of a record that ensures that it has not been changed or corrupted in any meaningful way, and "authentication is used to mean the verification that a record was indeed generated by its purported author or organization at the time and under the conditions of its purported generation. Our broad usage of "authenticity" is closer to the meaning of the word in general parlance: in this report, it denotes the full range of properties that establish the legitimacy of a record, including issues of integrity, completeness, correctness, validity, faithfulness to an original, meaningfulness, and suitability for an intended purpose. Authenticity issues are discussed in further detail elsewhere [18, 21].

[44] There are of course issues of context, provenance and other metadata, which are not part of the physical object itself, but all of these concepts apply to digital records in essentially the same way as they do to traditional records, so they will not be the focus of the discussion here.

[45] Such properties may not be relevant for strictly archival purposes. but they may be of great value for historical research, aswell as providing forensic evidence that may in some cases bear on their authenticity.

[46] The concepts of "vernacular renditions" and "vernacular extraction" are discussed in [24].

touching the originals. A digitized image of a record is another form of vernacular rendition, which allows users to read the record remotely over computer networks or (if the text of the record is captured, as well as its image) to search its content. Note that new vernacular renditions may have to be generated over time, since each era defines its own ideas of convenience and access; for example, digitized versions of records are vernacular renditions demanded by the current era. A given vernacular rendition may become a new historical object (or even a new record) in its own right, but it does not generally replace the original record.

4.4.2 Use of originals vs. surrogates

It is illuminating to examine the ways that various kinds of users of archives use originals versus vernacular renditions. Scholars may use original records directly, despite their relative inaccessibility. This may require traveling to archives, learning to read outdated scripts or dialects, learning any relevant conventions for typography, formatting, sequencing, etc. (for example, it is obviously important to know that Hebrew or Arabic books are to be read right to left). By assumption, scholars are able to invest the time and effort needed to learn to use such records in their original forms.

Students or more casual users may sometimes need or want to access original records as well. If such access is permitted, it may need to be mediated by scholars or archives staff, either to protect the original records or to interpret them as necessary for non-scholarly users. Some casual users may simply want to view original records to convince themselves that they exist or to experience the thrill of being in the presence of historically significant artifacts, like the Rosetta Stone or the Magna Carta. In some cases, non-scholarly users may need to invest some effort in learning the basic conventions embodied in these original records to be able to understand them at all; this effort will be far less than that required to become a serious scholar in the given subject area, but it may afford casual users a sufficient degree of access for their purposes.

In many cases, scholars themselves utilize vernacular renditions of various kinds as surrogates for original records. Scholarly research forms a spectrum that requires a range of different degrees of access to records. For example, if a scholar's research requires accessing only the textual content of a record, it may be sufficient to read a modern translation of the original[47]. In addition, surrogates allow scholars to juxtapose and compare records whose originals may reside in widely separated locations. Casual users or students may also satisfy their needs by accessing vernacular renditions of records in most cases. However, if these vernacular renditions are not to be misleading, they should ideally be generated directly from the original records by scholars who are sensitive to the question of which attributes of the original a given vernacular rendition should reproduce. Different vernacular renditions, intended for different purposes or different classes of users, may retain different attributes, and users of these vernacular renditions implicitly rely on scholars to ensure that these renditions are valid in the appropriate ways.

4.4.3 Digital original records

The concept of an original is far more problematic in the digital case. It is widely--and we believe persuasively--argued [26] that a digital artifact is a logical entity that is not

[47] Even art historians utilize reproductions for much of their research, despite the vast differences between the poor photographic reproductions in many art books and original works of art.

essentially related to the physical storage medium (or "carrier") that happens to hold it at any given time. This implies that a disk or other storage object that happened to hold a digital record when it was first created (or when it was acquired by an archives) should not be considered the original record. The logical bitstream that is stored on that (or any other) digital storage medium is a much more legitimate candidate for the original record--though as discussed above, this bitstream must be associated with an appropriate interpreter (and appropriate contextual metadata) in order to be meaningful.

The separation of the logical record from its physical carrier implies that there can be no single, unique instance of an original digital artifact. Any copy of a logical bitstream is identical to any other copy: since it is just the logical aspects of this bitstream that are meaningful, any copy has exactly the same attributes as any other copy, unlike a physical copy. It is in fact logically impossible to distinguish between two copies of a given bitstream [48]. It appears inescapable that the concept of a unique original must be sacrificed in the digital realm.

It may be tempting to eliminate the concept of an original in the digital realm altogether; however, this concept is too useful to discard lightly. We can define digital originals by analogy to traditional originals:

> **A digital-original is any representation of a digital informational entity that has the maximum possible likelihood of retaining all meaningful and relevant aspects of the entity** [49].

Ideally we would like to replace the phrase "has the maximum possible likelihood of retaining" in this definition by "retains" in order to ensure that a digital-original in fact retains all meaningful and relevant aspects of the entity; but this may not be possible, since even our best preservation approach may sacrifice or distort some aspects of the original entity [50]. So the best we can do is to define a digital-original as that which can in principle retain as many of its meaningful and relevant aspects as possible, just as a traditional original retains as many of its aspects as possible, depending on how well it is preserved physically.

By this definition, a digital-original cannot be just a copy of the original bitstream of an entity, since--as argued above--the bitstream by itself does not retain any of the meaning of the entity. A digital-original must include some mechanism for rendering the entity appropriately in the future [51].

---

[48] One might nevertheless try to argue that the first copy of such a bitstream given to an archives constitutes the original record, and that any subsequent copy is not the original; but this argument is flawed. The moment the original bitstream is stored on any storage device or processed in any way by a computer (even to transmit or display it), it will be copied, and there will be no way to tell which is the original and which is the copy. The very process of ingesting a record into a repository is one of copying its bitstream, making it meaningless to talk about the "original bitstream" of a record at its time of accession. A bitstream is a mathematical entity that has no original, just as there is no original number 5.

[49] A digital-original should always be immutable. The technical and procedural means of ensuring and verifying this immutability over time are beyond the scope of this paper, but we assume that sufficient safeguards of this kind will be made available.

[50] Similarly, it is impossible to guarantee that a traditional original will retain all of its meaningful and relevant aspects as it physically decays over time.

[51] Of course, the "meaningful" and "relevant" aspects of a record are in the eye of the beholder, and different users in different epochs or with different purposes may have different ideas of what is meaningful and relevant in a record. We may choose to transform a digital record from its original format into a simpler one that sacrifices some aspects of the original and arbitrarily define the simpler version to be the "original"

In Section 4.6, we will explain how emulation might be used to construct something that satisfies this definition. But for now we will assume that a digital-original record consists of a virtual recreation of the original computer system on which the original record was rendered, which runs the original software that rendered that record, so as to render it on some future computer, as illustrated in Figure 1. This allows us to explore various analogies between the use of traditional originals and the use of digital originals.

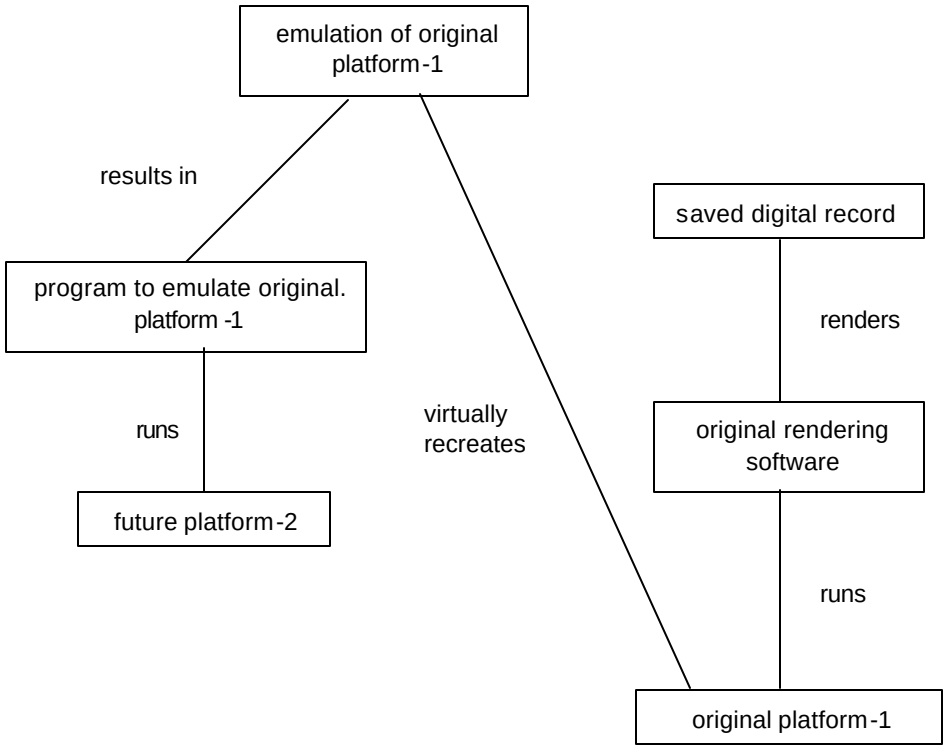

Figure 1: Using emulation to render original records.

Future scholars would be able to use a digital original record by running the original software that rendered that record on a virtual (emulated) recreation of the original computer on which that software ran. For example, a scholar in the year 2300 might view an original Excel 2000 spreadsheet displayed essentially as it was originally, on

record of interest; for example, we might transform a complex word processing record into a simple ASCII text file and define the ASCII version as the original record of interest, in which case we essentially define the text as the only meaningful aspect of the original record. The ASCII file—along with some mechanism for rendering ASCII--would then be the digital-original of the transformed record, though we would not call it a digital-original of the earlier version of the record.

what would appear to be a Microsoft Windows desktop, displayed on a year 2300 computer [52]. Of course we can only guess what a year 2300 computer will be like, but we assume that it will have some way of presenting two dimensional images, since such images seem likely to remain in use so long as the human visual system remains intact. These images might be displayed virtually as mid-air holograms or even projected directly onto the user's visual cortex, but we assume it will remain possible to view such images and that it will be possible to produce an emulated screen image whose effective size and pixel resolution falls within the range of our current computer screens [53].

Using emulation, our future scholar is able to view the Excel 2000 spreadsheet; but there are problems. For example, the initial display will show only the top-left segment of the spreadsheet. Seeing the rest of the spreadsheet requires scrolling, paging or issuing various commands to the Excel program. It may be far from obvious that scrolling in Excel is accomplished by pointing and clicking a pointer device (if the future system even has any such device) in the "scroll bar" or on one of the appropriate arrows in the display [54]. Therefore, even in order to see the entire spreadsheet, our scholar must learn the basics of how to use Excel [55]. In order to view formulae within the spreadsheet's cells or search for specific content, our scholar may have to study Excel in further depth [56]. This is exactly analogous to the need to understand the dialect or physical conventions of a traditional record. Serious scholars who need to work with year 2000 documents will presumably invest the time and effort to learn how to use Microsoft Windows and associated applications on an (emulated) Intel-based computer.

However, students or more casual users may want to access such digital-original records as well. Although these users cannot be assumed to have the time or

[52] It is impossible to guess how users will invoke programs to view documents in the year 2300, but whatever they normally do to cause this to happen, our scholar would do to invoke the emulation environment to run Excel 2000 on the desired spreadsheet. This might or might not involve anything remotely like double-clicking on an icon representing the spreadsheet in question.

[53] We do not assume that the physical display of the future computer has this size and resolution--it may be far higher resolution or even use technology for which size and resolution are irrelevant concepts. We merely assume that it is capable of emulating the correct size and resolution in some acceptable sense. Furthermore, most current computer programs are designed to run on a range of different computer systems, having a range of different display screens, processor speeds, memory capacities, disk sizes and speeds, etc. This has elsewhere been called the "native range of variability" of a program [23]. It is meaningless to ask what specific values of these system characteristics such a program required, since it made no such specific demands; rather, it is sufficient to ensure that a future emulation environment can create an emulated system that falls within this range.

[54] For example, it may be that the normal way of navigating on the year 2300 computer is to move and blink one's eyes rather than moving or clicking a manual pointing device.

[55] In some cases the emulation environment may be able to translate actions from the future vernacular interaction idiom into the required past idiom, e.g., allowing users to move and blink their eyes and automatically translating this into the point and click operations required by Excel 2000.

[56] This implies that it may be important to preserve user documentation for rendering software as well as the software itself; this would correspond to user manuals or guides for how to use the software, not detailed specifications of what the software does or how it works (which are often nonexistent). Fortunately, since user documentation for most programs is available in online form, preserving such documentation is again merely a matter of saving bitstreams. However, these bitstreams must be rendered properly in order for the documentation to be human-readable in the future, so software to render the formats of such documentation must be saved as well--along with any documentation required to use that software in turn. For example, in order to enable future scholars to use Excel 2000, we need to save Excel 2000 documentation, perhaps in PDF format; in order to enable future scholars to read this documentation, we may need to save Adobe Acrobat to render PDF, but we will also need to save documentation for how to use PDF, etc. This is a recursive problem, but the recursion can be ended in one of two ways: (1) either we can provide documentation in a simple textual format that is rendered by a simple program whose use is described in offline, human-readable text, or (2) our future computing environments can be made to automatically interpret a future user's request for documentation on obsolete rendering software as a request to run rendering software for the documentation itself. Underlying documentation of this sort was referred to as "bootstrap" information in [19] and is discussed under "Representation Networks" in [25].

inclination to study obsolete systems or software, there are two ways they might be served. The first is to include on the year 2300 computer a "help" system that explains how to use the emulation environment in general as well as each specific emulated system and obsolete application program[57].

An alternative way of helping future casual users is to generate digital "vernacular renditions" that are exactly analogous to the traditional surrogates discussed above. A digital vernacular rendition would be a digital surrogate in some future digital vernacular form, generated from the emulated digital-original record. For example, suppose the year 2300 computer has an application called DataView2300 that allows creating, displaying and manipulating three-dimensional holographic data spaces that constitute the future evolution of our notions of spreadsheets and databases. One vernacular rendition of the Excel 2000 spreadsheet might consist of a transformation of the spreadsheet into this DataView2300 format.

As illustrated in Figure 2, these mechanisms would provide users with two alternative ways of viewing a digital record: either in its original format or in a modern, vernacular format.

Just as traditional vernacular renditions should ideally be generated from original records, digital vernacular renditions should be generated from digital-originals. Ideally, the emulation environment on the future computer would provide facilities for generating (or "extracting") such vernacular renditions automatically or on demand[58]. Our casual user would therefore simply ask to see the Excel 2000 spreadsheet in DataView2300 format, and the system would retrieve or generate the required vernacular rendition and display it. The user could then search, excerpt or otherwise manipulate the data from the Excel 2000 spreadsheet in the familiar, vernacular environment of DataView2300, just as today's student might cut and paste text from a digitized textual transcription of the Rosetta Stone into a Microsoft Word document to incorporate it into a school paper. As in the traditional case, different digital vernacular renditions might be generated for different purposes or different classes of users; these might retain different attributes, and users of these vernacular renditions would implicitly rely on others to ensure that these renditions were valid in whatever ways were appropriate[59].

[57] As a minimum, this could include an overall explanation of how to use the emulated computer and operating system (e.g., how to open, close and scroll windows, how to run programs, etc. on an Intel-based Windows system), including an explanation of how to use the original system's native help facilities and documentation to learn how to use its programs. In addition, more specific help could be provided for how to use programs of common interest, which might include Excel 2000.

[58] The idea of vernacular extraction was suggested in [18, 22]. A recent variation on this idea by the CAMiLEON project is referred to as "migration on request" [14]; in order to minimize the chance of corruption, migration programs would be written before an older format becomes obsolete, and these programs would be run in the future under emulation. For further discussion of vernacular extraction, see Section 5.2.3.

[59] Generating the more elaborate of these vernacular renditions might require scholarly input, as is required to generate current vernacular renditions from ancient records or texts. The emulation environment should help scholars create such renditions when it cannot create them automatically.

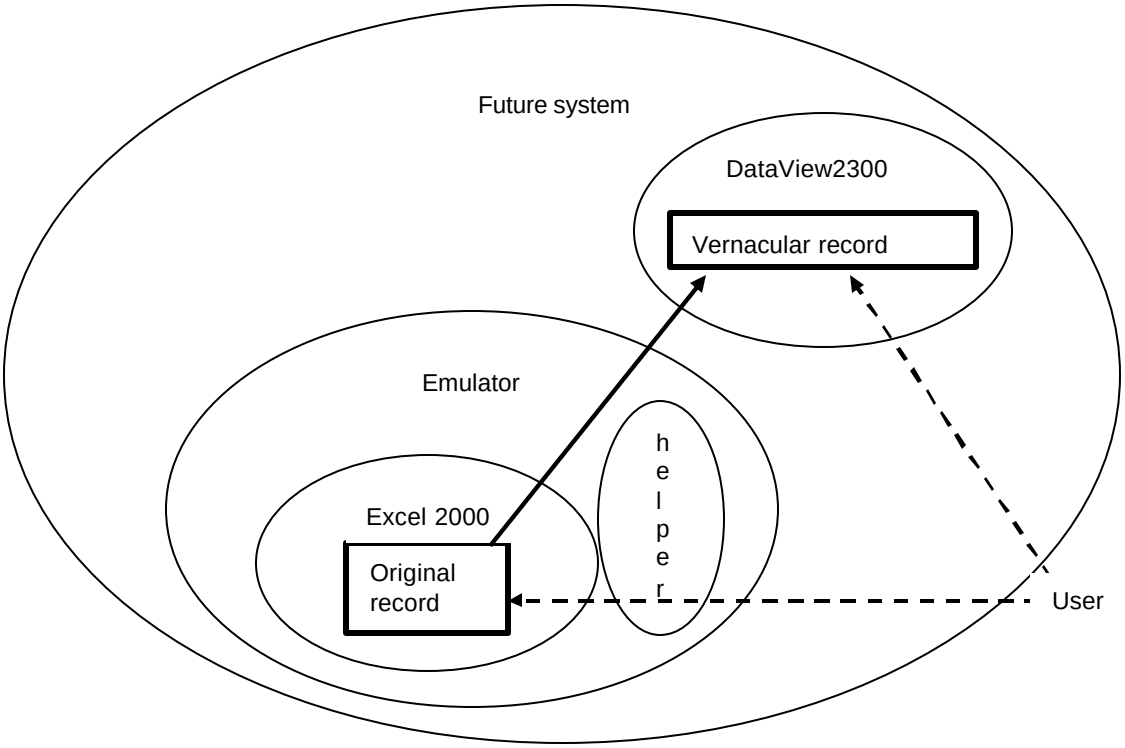

Figure 2: Accessing originals or vernacular renditions.

4.5 Emulation support for vernacular extraction

An emulation environment should incorporate built-in facilities to enable it to perform automatic vernacular extraction or to help scholars perform it. It is not yet clear exactly what these facilities should be, but future research should clarify this question. It is possible that an emulation environment can utilize its host hardware platform's input/output capabilities to capture rendered objects in a format that is native to the host platform. A simple example of this occurs when running emulators of old computers on modern systems. Whereas the old computer may have originally printed its output text on a teletype, the emulator may emulate this by displaying that text in a window that emulates the teletype's paper; as displayed on the host platform, this window contains ASCII text that can be highlighted with the host platform's pointing device and copied to the host operating system's "clipboard" as modern ASCII text. This text can then be pasted into a different window running some modern application on the host platform (for example, a window displaying a Microsoft Word document). This operation amounts to a simple vernacular extraction of text from the emulated system's teletype into a modern word processing document [60].

[60] Note that we assume that vernacular extraction is performed by accessing the digital-original by means of emulation. Although the original rendering software for a given digital record may include editing capabilities

Additional software may have to be written to perform vernacular extraction [61]. Ideally, as advocated in [14], such extraction software should be written while the original format of a record is still current, since the understanding of that format will begin to evaporate once it becomes obsolete [62]. On the other hand, there are limits to how much of this can be done in advance, since the needs of future vernacular formats will become apparent only in the future. Various kinds of generic extraction software might be written for each format while it is still current, for example, to extract its contents as standard text, image or other contemporary formats, in the hope that these will facilitate generating desired vernacular extractions in future formats. However, any such extraction software will itself have to be repeatedly ported to future computing platforms in order to be usable in the future, thereby creating an additional preservation problem [63]. This suggests that it may be worth pursuing the automatic (or semi-automatic) approach suggested above, in which the emulation environment itself would perform vernacular extraction in the future by utilizing capabilities of future host platforms.

4.6 Using emulation to preserve digital records

Given the above background, motivation and context, we can now examine in detail how emulation might be used to preserve digital records.

Referring back to questions (1) and (2) in Section 3.1 above, the emulation approach to preserving digital-original records consists of:

1) Saving the original bitstream of a digital record

2) Rendering the original bitstream in the future by running the record's original rendering software on an emulated recreation of the original computer platform on which that software ran.

The original rendering software for a given digital format embodies the definitive interpretation of that format. Running that software to render a record ensures that it will be rendered correctly (within the limits of emulation discussed below), while avoiding the need to transform the record in any way, thereby eliminating any chance of corrupting it. The original bitstream is considered immutable for all time and is merely copied bit-for-bit onto future storage media, as successive media become unusable or obsolete. The original rendering software is similarly treated as an immutable bitstream, which is similarly copied onto new media as necessary [64]. So far, this approach simply requires the ability to preserve bitstreams over time [65].

---

that can in principle change (and re-save) the record, this introduces no danger of corrupting the original, since the emulation environment will always be working on a copy of the immutable digital-original.

[61] Although original rendering software may provide various "export" options allowing the contents of original records to be transformed and saved in various other formats, all such formats will be contemporaries of the original format rather than future vernacular formats.

[62] Specifications for these obsolete formats can be saved to help write such extraction software in the future, but as argued above, many formats are not well documented, making it problematic to reverse engineer them in the future.

[63] Note, moreover, that this is another case where executable software must be preserved in executable form. Such extraction software may therefore best be preserved by means of emulation, following a similar argument to that developed here for preserving rendering software. This is also the solution suggested in [14].

[64] To simplify this discussion, the original operating system environment in which the rendering software ran will be considered a part of the overall rendering software suite, all of which can be treated as a single bitstream.

[65] Underlying this approach is the assumption that bitstreams can be preserved indefinitely. Although this is not a trivial requirement, ancient symbol streams (such as hieroglyphics) attest to the feasibility of this task, and many techniques exist for ensuring and verifying that bitstreams are copied and preserved perfectly.

Since all programs ultimately need to run on hardware, emulation is used to allow running the rendering suite on future hardware. Instead of trying to preserve physical hardware (which is recognized as infeasible [26]), this approach turns the original hardware into software, i.e., an emulator program that recreates the hardware in virtual form.

To summarize, saving a digital artifact in its original form requires saving all of the following:

1) The original file(s) that represent the artifact's bitstream

2) The original software suite that rendered it

3) The original hardware that ran all that software

All of these except (3) are bitstreams, which are relatively easy to preserve. Emulation reduces (3)to yet another bitstream, thereby eliminating the need to preserve hardware. However, for this to work, we must be able to run saved emulators on future computers.

Note that none of the original rendering software suite required by a digital record (including the operating system) is ever rewritten or examined: it is simply used indefinitely as a black box [66]. The only new programs that have to be written to make this approach work are hardware emulators. The key question becomes one of ensuring that we can run old emulator programs on unknown future computers.

4.7 Running emulators on unknown future computers

Although emulation may sound difficult, several factors work in its favor. First, as noted above, we emulate hardware, not software,. Because computer hardware must be well specified in order to be manufactured, it is generally easier to emulate than software–which often requires elaborate reverse engineering [67]. In addition, an emulator of a given computer can and should be written while that computer is still extant. This allows validating the emulator against the computer that it emulates; if each emulator is made immutable once it is validated, it should continue to remain valid indefinitely. Finally, there are relatively few computer systems in common use at any given time (at least in the recordkeeping context), and since only one emulator need ever be written for any given computer, relatively few programs must be written to implement this approach.

The challenge is to ensure that once an emulator program is written, it can be run on any unknown computer in the distant future. This in turn sounds more difficult than it may be. The key is to ensure that once a given emulator program has been written

---

Whereas it is no doubt impossible to guarantee that anything can be preserved perfectly and indefinitely, it should nevertheless be feasible to approximate this goal more closely for digital bitstreams than for any other form of information.

[66] The vast majority of software is used this way: if we needed to be able to modify programs or understand how they worked, they would be of little practical value.

[67] Although the detailed design and implementation specifications of computer hardware may be proprietary, its functional specifications must be public in order for it to be usable. Computers are machines intended to run programs, so they are generally usable (and marketable) only to the extent that their functional specifications are publicly and accurately available to those who write software for them. We hypothesize that it is precisely this level of functional specification that is needed to emulate hardware for preservation purposes.

and validated against the computer that it emulates, the emulator is never revised. That guarantees that the emulator's behavior can never change, which ensures that it will always faithfully emulate the computer it emulates: this in turn ensures that all programs that ran on that computer will continue to run correctly under the emulator indefinitely. But how can we ensure that an emulator program--even if it is never revised--will always execute the same way if it is run on different computers in the future? That is:

How can we ensure that a given emulator program will run correctly on any future computer on which we run it?

There are two aspects to this question:

i. How can we ensure that any future computer will be able to do whatever it needs to do to execute any emulator of any previous computer?

ii. How can we make our emulator programs run correctly on successive future computers without undue effort?

The answer to (i) requires a leap of faith--but only a small one. The issue is whether all future computers will be able to perform (either in hardware or software) all of the logical functions that their predecessors performed. Although it may at first appear unlikely that all future computing paradigms–such as quantum computation--will subsume the capabilities of current computers, this is actually quite plausible, since these capabilities are founded on simple, universal mathematical and logical operations, whose future utility seems certain no matter what new capabilities are added to them [68]. The extension of this argument into the far future is slightly more speculative: it requires assuming that any new capabilities that are introduced (for example, quantum computation itself) will similarly be so universal and so useful that they will in turn be carried over into their successor computers, even though those successors may use yet another different paradigm. Actually, this is a reasonable assumption, since any future paradigm that is long-lived enough to be used for digital records will by definition be long-lived enough to be used in the development of a significant number of programs and programming techniques. Therefore, even if this paradigm is eventually superseded by a different one, it will presumably be worth subsuming its capabilities in the new paradigm to avoid having to discard and recreate all of the programs and programming techniques that it spawned [69].

If we assume that future computers will be able to run all of the old emulator programs that we write, we must still answer (ii), i.e., how can we get our emulator programs to run correctly on each generation of future computer? Two general approaches to this problem have been proposed [24]: Chaining and Rehosting. In addition, an intermediate approach has been discussed in [22], which makes use of an Emulation Virtual Machine. The first two of these are discussed briefly here, after which the last is elaborated in further detail.

[68] This is more likely than that future application programs will subsume the behavior of current digital formats.

[69] This argument is more speculative when applied to the application program capabilities associated with digital record formats. That is, it seems more likely that mathematically-rigorous computational methods will be carried over into future computers than that the ad hoc features of specific digital formats will be carried over into future software. Indeed, incompatible formats have been the norm, whereas computational capabilities have remained largely unchanged throughout the history of computer science.

4.7.1 Chaining

Chaining is an approach which allows each emulator of one computer to run indefinitely once it has been implemented on only one other (successor) computer: if the successor computer is later emulated on its own successor computer, any previous emulator can be run under a chain of emulators, as illustrated in Figure 3. For example, for a sequence of computers C1, C2, C3, when C2 replaces C1, we write an emulator 1E2 that runs on C2 and emulates C1, thereby allowing it to run any program that ran on C1. When C2 is in turn replaced by C3, we then write an emulator 2E3 that runs on C3 and emulates C2, thereby allowing it to run any program that ran on C2. In order to emulate C1 on C3, we then run 1E2 under 2E3[70]. This approach is conceptually the simplest and requires writing the fewest programs, but running one emulator under another this way is relatively inefficient, since each emulator runs significantly slower than the computer (or emulator) that executes it.

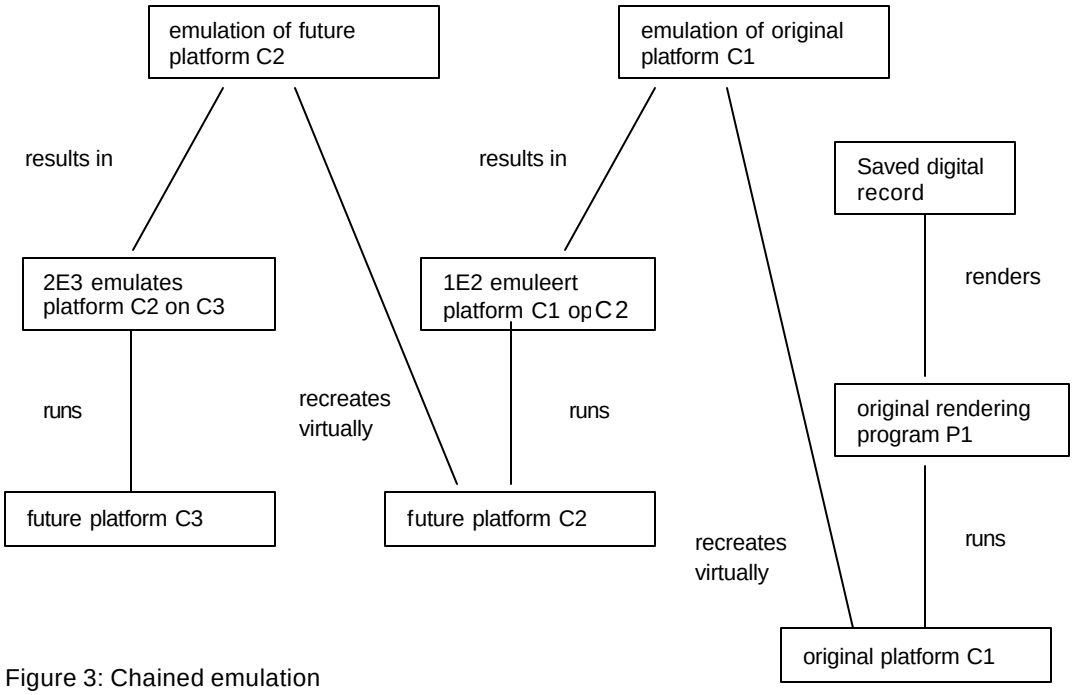

Figure 3: Chained emulation

Despite the fact that its inherent inefficiency may make chaining unattractive as a general approach, its simplicity make it useful in several situations. First, chaining provides a way of continuing to run existing emulators of older machines, without having to rewrite them to run on future computers. For example, emulators of many classic computers currently run on the ubiquitous Intel platform, so writing a single

[70] 1E2 is a program that executes on C2 and runs programs intended to execute on C1. Similarly, 2E3 is a program that executes on C3 and runs programs intended to execute on C2. Since 1E2 is a program that executes on C2, and since 2E3 runs programs that execute on C2, 1E2 will run under 2E3. An original program P1 that executed on C1 can therefore be run under 1E2 when 1E2 is run under 2E3 on C3. Similarly, when C3 is eventually replaced by C4, we write emulator 3E4 and run P1 under 1E2 under 2E3 under 3E4 on C4.

emulator of a generic Intel platform would allow running all of these classic emulators in the future under the Intel emulator, without rewriting any of the older emulators. This "opportunistic chaining" would allow such existing emulators to run indefinitely with no additional effort. Similarly, chaining can provide a default or fallback mechanism for ensuring that any emulator, once written, can be run indefinitely under a chain of future emulators, regardless of what other mechanisms (such as rehosting or an Emulation Virtual Machine) may also be employed. Finally, as suggested below (in Section 4.7.3), chaining offers one way of ensuring the longevity of Emulation Virtual Machines that are themselves superseded by newer ones.

4.7.2 Rehosting

As an alternative to chaining, we can rehost emulators on successive future platforms. To do this, we first write an emulator 1E2 of computer C1 and initially run 1E2 on computer C2 (the same as in chaining). When C2 is later replaced by C3, we then rehost 1E2 on C3, producing 1E3: that is, we make our emulator of C1 execute directly on C3 (as illustrated in the bottom left corner of Figure 4) instead of running it under another emulator. The rehosted emulator 1E3 would run more efficiently on C3 than a chained emulator 1E2 running under 2E3, but we still need to write emulator 2E3 in order to run rendering software that originally ran on C2 (as illustrated in the top left corner of Figure 4), so this requires more programming effort. That is, we would have to rehost(or "port") every past emulator to every new computer that comes along, as well as writing a new emulator for every new computer that becomes obsolete. Some advocates of this approach [10]suggest writing all emulators in a simple, higher order programming language whose semantics would be kept stable over time to help ensure that each new porting of an old emulator would work correctly. Nevertheless, rehosting risks undermining one of the key advantages of emulation (mentioned above), that each emulator can be made immutable once it is validated. If each emulator must be rehosted on each new computer, then we are essentially reimplementing each emulator each time we rehost it, which can greatly reduce the confidence in its validity.

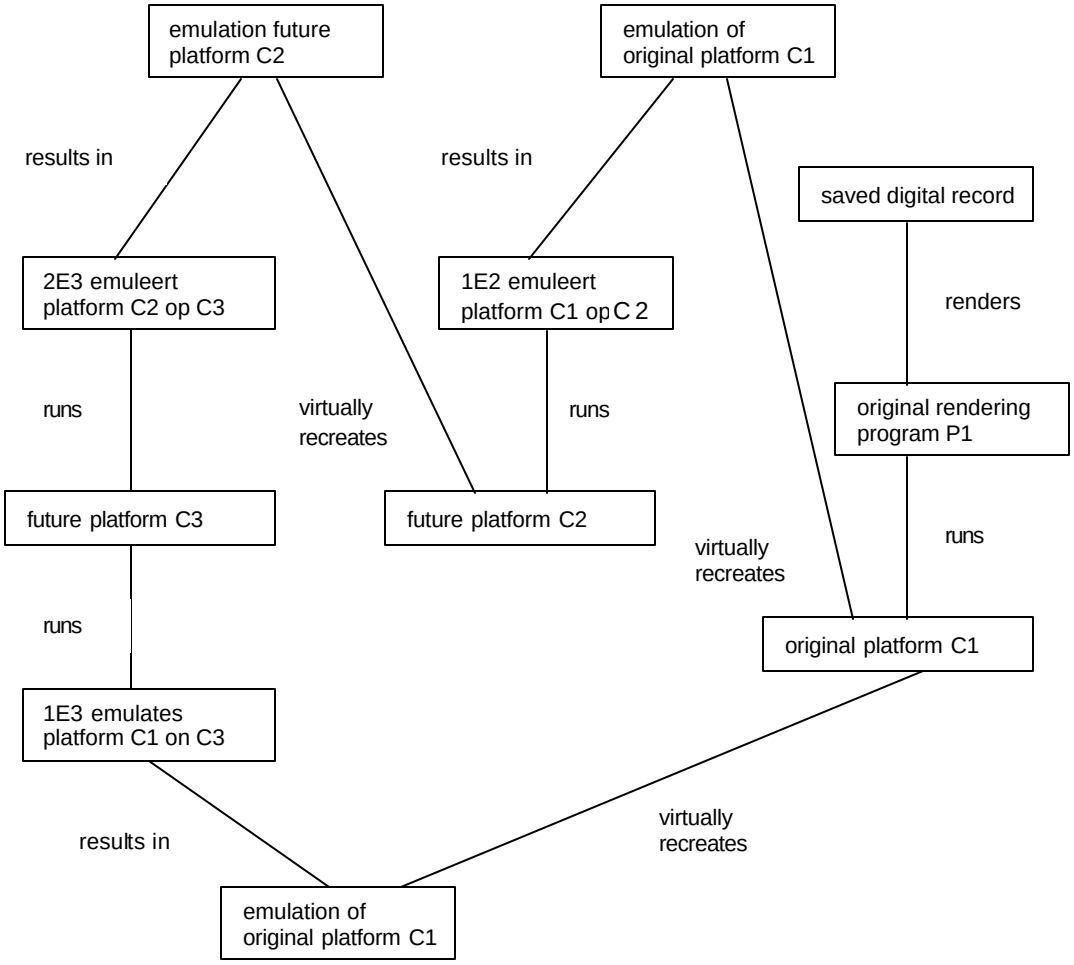

Figure 4: Rehosted emulation 1E3 emulates platform C1 on C3

4.7.3 Using an Emulation Virtual Machine (EVM)

An intermediate approach that offers a number of advantages is to run all emulators on an "Emulation Virtual Machine" (EVM) [71]. The concept of a virtual machine (VM) is very similar to that of an emulator. A VM is a program that runs on some real computer and thereby virtually creates some imaginary computer [72]. A VM does essentially the same as an emulator, except that an emulator emulates some other real machine, whereas a VM typically implements a computer that has never existed

[71] See note 33.
[72] A VM typically implements an imaginary machine rather than a real one to free it from the constraints of physical hardware. One of the current familiar uses of a VM is to run Java programs: these are written to run on the Java Virtual Machine (JVM), which is in turn hosted on many different real computers.

as hardware [73]. We usually say that an emulator "emulates" another computer, whereas we usually say that a VM "is implemented" on a computer; but the concepts are essentially equivalent.

One of the chief motivations for creating any VM is to provide an easy way to run a given set of programs on a wide range of different computers. In order to do this, programs are written to run on the VM rather than on any specific computer [74]. If the VM is then implemented on many different computers, all of the programs written for the VM can be run on any of those computers. This is very similar to the emulation approach, in which writing a single emulator program allows any rendering program that ran on the emulated computer to run under that emulator on any computer that can run the emulator.

A VM can greatly simplify the emulation approach. The idea is to define an EVM which becomes the virtual platform on which all emulators are written to run. As each computer in a given epoch becomes obsolete, an emulator of that computer is written to run on the EVM. As each new computer is introduced, the EVM itself is then implemented on (i.e., ported to) that new computer, as illustrated in Figure 5. For simplicity, the figure shows the EVM running only an emulator of platform C1, but the EVM would also eventually run emulators of C2 and C3, in order to render digital records that were created by software running on those platforms, as well. Once the EVM runs on a given computer, all previously written emulators will run on that computer, since all emulators run under the EVM. Therefore, no emulator need ever be rewritten or rehosted on a new computer: each emulator always and only runs on the EVM. The only programming effort involved in this approach is:

Before any old computer becomes obsolete, an emulator of it must be written to run on the EVM [75]

When any new computer appears, the EVM must be ported to it [76]

[73] It is not strictly necessary that a VM represent an imaginary computer, in which case an emulator and a VM can be said to be the same thing.
[74] Returning to the Java example, this allows Java programs to be independent of specific computer hardware platforms.
[75] Actually, emulators need only be written for old computers that serve as the only way to run rendering software for digital formats that are used for archival records. This may be a very small subset of all computers. For example, if all of the rendering software for a given set of record formats runs on many different computers, it is not necessary to emulate all of them.
[76] Similarly, it is not necessary to port the EVM to every new computer. It is logically sufficient to ensure that there is at least one type of computer available in each successive hardware epoch that can run the EVM to access old digital records. Note that there are typically many individual computers of any given type (for example, there are currently millions of Pentium-based computers), so porting the EVM to a single type of computer--especially if that type is chosen as one that is in common use--will allow accessing digital records on many distinct machines in many different organizations and locations. To provide additional redundancy, the EVM may be ported to several types of computers in each epoch.

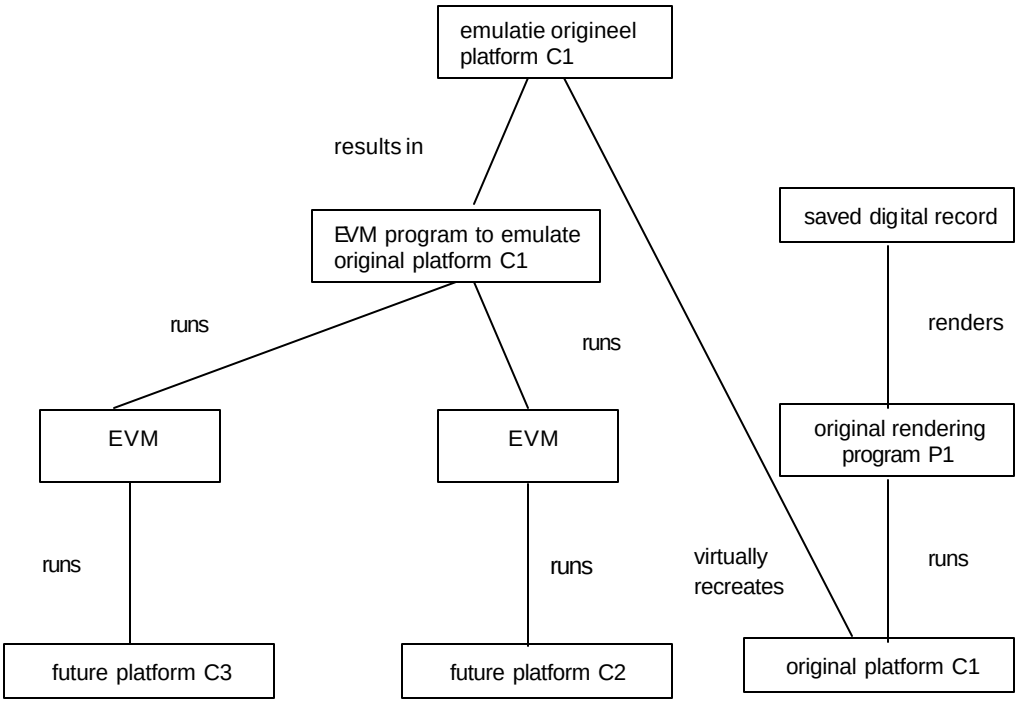

Figure 5: Running emulators on an EVM

An ideal EVM would be designed to have two essential properties:

a) It would support a wide range of emulators

b) It would be relatively easy to implement on any computer

The first criterion (a) would be satisfied by providing a rich input/output (I/O) environment and by making the EVM customizable in a number of ways (e.g., allowing it to be run at various clock speeds, allowing it to vary its display and other I/O characteristics, etc.) The second criterion (b)would be satisfied by designing the EVM to be based on a relatively small number of "primitive" functions: these and only these functions would have to be implemented on a new computer in order for the EVM to run on it. In addition, an EVM should be designed to have well-defined semantics, in order to allow the behavior of its implementations to be validated [77].
One of the key advantages of an EVM is that it could be designed with emulation and preservation in mind. As discussed in the next section, an existing VM is unlikely to serve as an ideal EVM, because its design may be constrained by other considerations [78]. Nevertheless, even if an EVM is designed specifically to support

[77] See Section 5.2.2 for further discussion of validation.
[78] There are many tradeoffs between designing an EVM specifically as an EVM versus using a more widely-supported virtual machine. The EVM approach presented here is not dependent on developing a dedicated EVM but merely on having a suitable VM that can serve the function of an EVM.

preservation via emulation, that design is unlikely to last forever. Programming styles and paradigms will evolve over time, leading to a desire to write new emulators using the newest paradigm, which the EVM may not support. More importantly, future computers with unimaginable new capabilities will be produced and will ultimately become obsolete, requiring emulation of those capabilities, which the EVM may not be capable of supporting. If the preservation community is in control of the EVM design, it may be able to resist such evolutionary pressures for a decade or two at a time, but they will eventually (and repeatedly) become irresistible. The obvious solution to these problems is to design a new EVM (which we denote EVM2) in the future, when the original EVM becomes outdated. When EVM2 itself becomes outdated, we would repeat the process, designing EVM3, etc.

But how will we run all of our old emulators (which were written to run on EVM) if we replace EVM by EVM2? It would seem that we would need to continue to run all of our existing emulators under EVM on future computers while running all new emulators under EVM2 on those same future computers. This would require porting EVM as well as the new EVM2 to every new computer, which would double our porting effort. And when we eventually introduce EVM3, we would wind up with three times the porting effort. Fortunately, there are two fairly straightforward solutions to this problem.

First, we can require that EVM2 be backward compatible with EVM in the sense that it guarantees that it will correctly run any program that ran on EVM. That is, we could add new features or implement a new paradigm in EVM2, but we would design it so that it would continue to support the features and paradigm of EVM. New emulators for new computers could be written to utilize the new features of EVM2, but all old emulators would continue to run on EVM2 without modifying them. As EVM2 itself becomes obsolete in the more distant future, EVM3 could be designed to be backward compatible with EVM2 (and therefore with EVM). This approach would ensure that all old emulators continue to run on all future EVMs [79].

Alternatively, we might use chaining (as described in Section 4.7.1 above) to enable emulators written for EVM to run on EVM2, EVM3, etc. The idea here is that when EVM is to be replaced by EVM2, we write a single new emulator—of EVM that runs on EVM2. Using the notation developed above, we will call this emulator 1EVM2. After having written 1EVM2, we can run all of our original emulators (which require EVM) under 1EVM2 on EVM2. When EVM2 is in turn replaced by EVM3, we write 2EVM3. We then run all emulators that require EVM2 under 2EVM3 on EVM3, and we continue to run emulators that require EVM under 1EVM2 which we now run under 2EVM3 on EVM3, as illustrated in Figure 6 [80]. This chaining solution is more attractive here than it was in the case of simple emulators, since efficiency should not be nearly as much of a factor. If we assume that each EVM will last a relatively long time (say, 10 or 20 years) compared to a physical computer, then the length of this emulation chain should grow relatively slowly compared to the expected increase in computer

---

[79] Note that this is somewhat the inverse of the rehosting approach for emulation above. That is, we make old emulators run on future EVMs, but rather than rehosting those old emulators by rewriting them to run on new computers, the approach here is to constrain the new (virtual) computers represented by EVM2, EVM3, etc. to be backward compatible with EVM, so that emulators never need to be rewritten. This is possible in this case (whereas it is not possible in the rehosting case above) because the succession of EVM designs is assumed to be under the control of the preservation community (whereas that community cannot be assumed capable of constraining the evolution of future hardware platforms).

[80] Of course, in all of these cases, the final emulator in our chain will execute on some physical computer. That is, EVM would originally execute on C1, EVM2 would originally execute on C2, and EVM3 will execute on C3.

power, making it feasible to run each EVM under emulation on the next EVM in the chain without undue loss of performance.

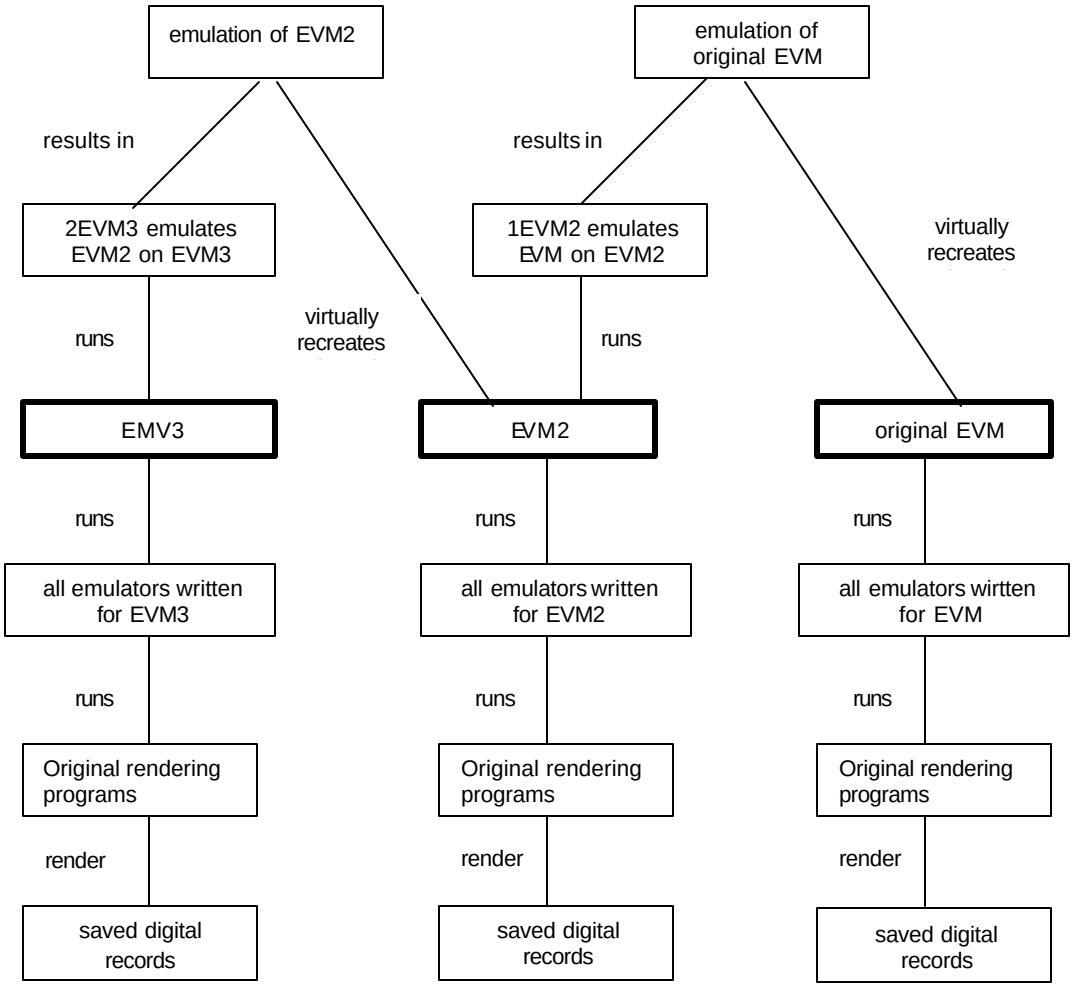

Figure 6: Chaining a sequence of EVMs

4.7.4 Existing candidates for an EVM

A natural question that arises in the context of the EVM is whether the Java Virtual Machine (JVM) could serve as an EVM. The answer is that it could, but it is not ideally suited to the purpose for a number of reasons. We discuss this question to shed further light on the EVM concept.

There are a number of ways in which the JVM is an excellent candidate for an EVM:

- It is quite popular
- It has already been implemented on a wide range of computers
- It includes a rich platform-independent I/O environment

However, it also has some significant disadvantages:

- Its continuing rapid evolution makes it relatively unstable
- It is language-specific
- It is relatively difficult to implement on a new platform
- It has no built-in support for vernacular extraction

By utilizing the JVM as an EVM, the preservation community could take advantage of the popularity of Java in the general computing community and the ubiquity of implementations of the JVM on a large number of platforms. Furthermore, Java supports most modern input/output capabilities, which should make it relatively easy to emulate most computers and peripherals in Java.

However, Java's ongoing evolution makes it less stable than it might be. In particular, much of Java is defined in class libraries that continue to evolve and spawn new variants. Even some of Java's fundamental user interface concepts and techniques continue to change as more experience is gained using Java. Although much of this volatility is restricted to the Java class libraries and language, some of it affects the design of the JVM as well.

Java's wide market acceptance is a two-edged sword. On the one hand, it results in many implementations of the JVM on many different computers, and it gives the JVM a critical mass of support, which makes it likely to survive for some time into the future. On the other hand, the market exerts considerable control over the JVM's evolution, making it relatively difficult for the digital preservation community to influence it.

The Java-specific nature of the JVM is also a major drawback, since although new emulators certainly could be written in Java, the restriction to Java makes it more difficult to adapt existing emulators that were written in other programming languages and environments [81].
The JVM is designed to be relatively efficient, to enable Java programs to perform competitively compared to programs written in languages that compile to machine code for their respective platforms. However, optimizing the JVM for performance has (among other factors) led to a design that requires a relatively big effort to port the JVM to each new platform. Porting the JVM typically requires implementing hundreds of distinct primitive functions. This is not necessarily a major issue so long as Java is used for purposes other than preservation that motivate (and pay for) porting the JVM to new platforms. Yet preservation must maintain a long-term perspective, and when

[81] Although it is of course possible to compile other programming languages into Java or into Java "byte code" (the pseudo machine code that Java compilers produce, which runs on the JVM), this does not provide straightforward access to Java's extensive class libraries, thereby limiting the potential for taking full advantage of the Java environment.

Java eventually falls into disfavor and disuse (as all programming paradigms do), these other nonpreservation uses of the language may no longer bear this cost, at which point it will have to be borne by the preservation community. In contrast, an ideal EVM would be designed to minimize the effort of porting, if necessary by de-emphasizing the need for high performance[82].

Finally, the JVM provides no inherent support for vernacular extraction, as discussed above. This is not surprising, since the JVM was not designed specifically for emulation or preservation purposes. Furthermore, as explained above, it is an unanswered research question as to what kind of support an EVM should provide for this. Yet because the JVM is controlled by market forces that originate outside the preservation community, it is not ideally suited to provide such support in the future [83].

IBM's Universal Virtual Computer (UVC) also has some advantages and disadvantages as a candidate EVM. Since it is still in an early stage of development, it may be premature to evaluate. It lacks any I/O capability, making it inadequate as an EVM. Nevertheless it is designed to be very easy to port, and since its design is not yet controlled by market forces, it may be more amenable to the inclusion of facilities to support vernacular extraction if such facilities are defined in the future. Furthermore, the "data extraction" approach that has been the main focus of the UVC work until now provides a form of migration on request, which is directly relevant to vernacular extraction [84].

4.7.5 Universal Virtual Computer

An emulation approach that uses the UVC differs somewhat from the original emulation concept. An emulator must still be written, but in this case it is for a non-existent, virtual computer, called the UVC (Universal Virtual Computer). The UVC is a computer with such a simple architecture and basic set of instructions that any software developer in the future will be capable of writing an emulator for the UVC. The UVC is then used to run an application (UVC data format decoder) that takes the original record as input and delivers a Logical Data Description (LDD) as output for the data. This logical data description is built up of tags that provide additional information about the content of the digital record. The additional semantic information is set up in such a way that, in the future, people will be able to interpret the logical data description without additional resources. After that, a viewer built in the future processes the logical data description, which displays the authentic digital record on the screen.

The Universal Virtual Computer preservation strategy is a variation on the emulation strategy and needs to be applicable in every future computer environment for complete preservation efficiency. The strategy only partly relies on emulation and contains some aspects of the migration strategy. Using the UVC, original data files are converted into a Logical Data Description (LDD) via a program written in the UVC programming language. This LDD is an independent, self-descriptive and clearly

[82] Of course it is desirable for any VM to perform well, but since the EVM will by assumption be run on future computers, their increased performance should more than offset any reasonable compromise in performance that the EVM has to make in the interest of increased portability.
[83] It is clearly unrealistic to expect any programming environment to support an undefined set of future requirements. However, it may be possible to define some generic emulation environment features to support vernacular extraction based on a relatively modest investment in research into this subject, after which such features could be designed into an EVM if it did not already provide them (and if its design were open to modification).
[84] See note 34 above.

structured data format that contains all the information needed for re-assembling the digital record in the future.

UVC data preservation
'Data preservation' is the first and simplest implementation form of the UVC strategy. In it, the data – the original file in its original format – is stored with a program that extracts the data out of the bit stream and describe this data simply and independently, so that a viewer can process the data.

The original file – for instance a JPEG file – is stored together with the specific UVC data format decoder program for JPEG. In the future this UVC JPEG program will be run on the UVC emulator. The UVC JPEG program reads the bit stream of the original file and produces an LDD as output (Logical Data Description). The LDD is reproduced on a future computer platform using a viewer that can be developed in the future based on the LDD Schema.

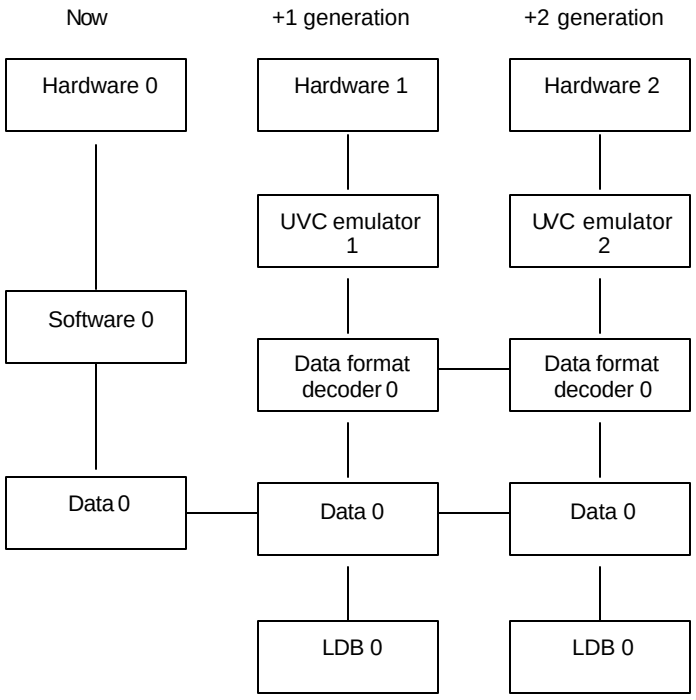

Figure 7. Diagram of the Universal Virtual Computer

The original bit stream is not changed in this strategy and the new file (the LDD), made when running the UVC data format decoder program, is not saved. The LDD is displayed by way of a viewer. The format and the structure of the Logical Data Description are so clearly defined that designing and writing a new viewer should be straightforward. If necessary, new viewers can be developed for future computer platforms.

At present, a separate viewer is needed for each type of LDD. This means that possibly hundreds of viewers are necessary. In practice, the number of different formats accepted by the Archival institutions might be limited by means of regulations or guidelines [85]. In the next phase of the UVC development, classes of objects will be formed that behave according to the same logic. A class of objects like this (for example, comprised of files in different image formats) will produce one LDD, for which only one viewer will have to be developed. It will, however, still be necessary to develop an individual UVC data format-decoding program for each of these file formats.

A disadvantage of the UVC emulation approach is that UVC data format decoder programs have to be written for each file type (to generate the logical data description). In addition, a new UVC emulator must be written for each new

[85] In the Netherlands the "Regeling Geordende en toegankelijke staat archiefbescheiden" (Feb. 2002) limits the number of file formats to be transfered to archival institutions.

generation of hardware that differs so much from previous generations that the old UVC emulator can no longer reliably run on it

In view of the wide variety of file formats and types of digital records, large numbers of decoder programs will have to be developed as quickly as possible, if the UVC is to be a feasible and workable strategy for the long term preservation of different types of digital records. The ultimate success of the UVC strategy is partly dependent on the extent to which this strategy is accepted by the software and computer sector. Software suppliers would have to develop a UVC data format decoder programme for their software that can make a logical data description based on the original file. When that happens the UVC strategy could expand enormously.

5 *Evaluation of emulation*

5.1 The potential of emulation-based digital preservation

If emulation can be made to work over the long term, it offers a way of preserving digital-original records. It would:

- Preserve digital records in their original forms
- Preserve executable records as executable
- Offer near-zero incremental cost for each format to be saved
- Offer essentially zero incremental cost for each individual record to be saved
- Offer a single, consistent preservation approach that could be applied universally
- Make it unnecessary to abandon records in obscure formats

The near-zero incremental cost of saving each additional format is aresult of the fact that no significant work need be done to add a new format to the emulation approach. The format in question need not be understood, reverse engineered or even documented [86]; it is merely necessary to save the rendering software for that format, which is just one more bitstream—and which is likely to be far smaller than the bitstreams of the collection of records that use that format [87]. The zero incremental cost of saving each
additional record is a result of the fact that records need not be transformed or processed in any way either initially or over time; it is merely necessary to save their bitstreams [88].

The universality of the emulation approach derives from the fact that any record in any format is preserved in exactly the same way, i.e., by saving the rendering suite for that format along with the bitstream of the record [89].

Its near-zero per-format cost and universality make emulation an attractive default or "catchall" preservation approach, as well. Regardless of whatever other preservation approaches are used for whatever types of records, emulation could be used in addition, to provide a backup for these other approaches and a way of preserving records in any obscure formats that the other approaches may not be able to handle cost-effectively. Since no other approach attempts to preserve access to records in their original forms, emulation would serve as a backup preservation mechanism that would provide access to the digital-original form of each record, which might be of great value if the primary preservation approach for a given record fails to preserve the behavior, appearance, or other desired aspects of the record.

---

[86] As noted above, writing software to perform vernacular extraction may require understanding the original format of a record; this may in turn require preserving some form of documentation or specification for that format. Nevertheless, no such understanding or documentation is needed to preserve digital-originals using the emulation approach.

[87] Note, however, that user-level documentation may have to be saved to enable future scholars to use rendering software, as discussed in note 49 above.

[88] Every digital preservation approach requires saving some sort of bitstream.

[89] This of course assumes that a suitable emulator for a suitable hardware platform is also created and is made to run on any future computer platform by one or more of the mechanisms described above. However, the creation of such emulators and the work required to make them run on future computers is a per-platform cost that is shared by all of the formats and all of the records that use the emulation approach.

5.2 Unresolved questions about emulation-based preservation

Emulation is a accepted technique that has been used for decades—often for preservation purposes, to avoid the need to rewrite old software to run on new computers. Nevertheless, it has not yet been systematically applied to the preservation of obsolete digital formats in the sense discussed here. Several factors seem to have contributed to the cautious attitude toward emulation.

First, emulation involves a number of remaining technical questions and issues which, as discussed below, require further research. This research requires a depth of computer science expertise that may be lacking in most archives and other organizations concerned with preservation; yet the computer science community has not been forthcoming in performing such research, perhaps because it does not see preservation as part of its domain.

Second, most records that have been produced to date are not inherently digital and so do not demand the preservation of their appearance and behavior that are offered by emulation. That is, simple text or page image records may be adequately preserved (at least for most purposes) by less extreme approaches.

Third, there are the practical questions of who would write emulators, who would design and port an EVM, etc. The impression exists that such programming efforts is inordinately difficult or expensive, and limited experience in producing preservation-oriented tools of these kinds makes it difficult to replace this skepticism with empirical facts. Additional unresolved policy issues surrounding the licensing of rendering software and operating systems and related intellectual property questions (as discussed in Section 5.2.9 below) also increase skepticism.

In addition, the emulation approach is often misinterpreted, for example by confusing software-emulation-of-hardware with operating system emulation (as mentioned in Section 4.2 above).

A number of authors have criticized the emulation approach. Bearman [1] takes the line that emulation attempts to preserve information system functionality rather than records; however, inherently-digital records may embody such information system functionality. Thibodeau [28] questions the efficacy of emulation but proceeds from emulation of the operating system. The OAIS [25] discusses emulation briefly, emphasizing a number of the concerns discussed here. Granger [6] as well has summarized many of the issues surrounding emulation.

It has also been said that emulation is another form of migration, since emulators may themselves have to be migrated to run on future computers. This is true for some implementations of emulation.[90] However if something like an EVM is used to enable emulators to run on future computers, then it is the EVM rather than the emulators themselves that must be ported to new computers over time. Whereas such porting is indeed a migration effort, as pointed out above, an EVM can and should be designed to make this porting straightforward, e.g., by implementing the EVM by means of a relatively small number of well-defined primitive functions. It should therefore be simpler to port a single EVM to a new computer than to migrate an exponentially growing number of records in different and/or complex formats.

[90] Actually, this is only applicable to a subset of the approaches to making emulators work on future machines. For example, if chained emulation (as described above) is used, then a new program must be written to emulate each computer on its successor, which involves a new programming effort rather than migration.

Nevertheless, in order for emulation to become viable as a preservation approach, a number of issues must be investigated further.

5.2.1 How well can emulators recreate old computer systems?

We have stated above that an emulator of an old computer system should be able to run any program that originally ran on that system. This is the basis for being able to render digital records in their original formats in the far future. Yet it may not be the case that any emulator can run any program that ran on the emulated system. Some programs may intentionally or inadvertently utilize subtle or even anomalous aspects of their original hardware platforms, such as the timing of individual operations or memory access. Unless an emulator faithfully recreates all such aspects of the original hardware's behavior, it is possible that some such programs will not run correctly (or at all) under emulation. In practice, most programs run under emulation without any trouble, but occasional problems do arise [91]. Although this does present a challenge, it may not be a serious one in the case of using emulation to preserve digital records, since such records are unlikely to have been usable in their original forms if they depended on subtleties of the platforms on which they ran. Another way of saying this is that if the "native range of variability" (as defined above) of a rendering program was large enough for it to have been usable in its original record keeping context, then it is unlikely to have depended on obscure details of any particular hardware platform.

Assuming that emulators can be developed that can run original rendering software, a number of aspects of hardware behavior must still be faithfully reproduced by an emulator in order to ensure that the rendering software for obsolete formats produces appropriate results when run on future computers. For example, if display resolution improves drastically in the future, the original appearance of low-resolution images must be retained. If sound is included in a record, its original fidelity must be reproduced. If animated or live-action images are included, their original frame rates and smoothness must be recreated. Similarly, the original speed of interaction and display generation of a record must be retained [92]. And the "feel" of any input peripherals (such as keyboards or pointing devices) may have to be recreated in order to preserve an authentic sense of what it was like to interact with the original record.

That is to say, an emulator must faithfully recreate all relevant aspects of its target system, such as execution speed, display resolution, refresh rate, brightness or color, display movement, sound quality, calendrical issues, access speed (and other characteristics) of its original storage devices or media, etc. Further research must be conducted to enumerate all relevant characteristics of this sort and to devise ways of reproducing them faithfully under future emulation [93].

---

[91] This has been particularly true of game programs that tend to access low-level features of their hardware platforms and may depend on timing or other subtle aspects of system behavior.

[92] In many cases, an emulator running on a future computer may run far too fast to faithfully recreate the behavior of original software and may have to be artificially slowed to match the execution speed of its original target computer. For example, an early word processing program (named WordVision) for the original IBM PC apparently used the computer's processor speed to control the repeat-rate and the delay between pressing and holding down any key on the keyboard and causing that key to repeat. When run on later, faster versions of the PC, this program became impossible to use, since merely touching any key would cause it to repeat an indeterminate number of times. A similar problem would occur if this program were run under an emulator of its original computer–if that emulator were allowed to execute at the native speed of a fast host computer.

[93] Existing standards may provide adequate solutions to some of these problems. For example, International Color Consortium (ICC) color profiles, based on CIELAB device-independent color spaces (CIE 1931, the Commission Internationale de l'Eclairage) can be stored with image files, enabling future emulators to recreate the original color of those images with a degree of fidelity at least equal to that obtained when those images were displayed on different computers within the native range of variability of the original

One area of emulation that is likely to require compromise is that of I/O modality. As discussed above, future displays may not look anything like our current two-dimensional screens, and future input mechanisms may not correspond to our current pointing and clicking devices. Since one of the fundamental ideas of using emulation for preservation is to avoid having to preserve or recreate physical hardware, it is not attractive to require the recreation of obsolete I/O devices (such as old CRT screens, keyboards, or mice) in the future [94]. We assume instead that virtual recreations of such devices will normally be used, utilizing whatever I/O modalities are native to the future computers on which our emulators run. Examples of this are common in current emulators of old computers: teletype printers are emulated by windows of scrolling text on modern display screens, old oscilliscope displays are emulated by modern graphical displays, and special-purpose buttons or switches are emulated by pictures of those items displayed on a modern screen and activated by pointing at them and clicking with a modern mouse. Analogous solutions will undoubtedly be necessary in the future [95].

5.2.2 Validation

Validating any approach to the preservation of digital records is problematic. If a digital preservation approach transforms original records without attempting to preserve them in their original form [96], the best that that approach can offer is "stepwise" validation, where each transformed version of a record is validated against the previous version. Only the first such step can involve validation against the original record, since the original is assumed to become unusable after its first transformation. Can emulation-based preservation be validated any more meaningfully than this?

There would appear to be several possible levels of validating emulation-based preservation:

I. Validate that an emulator faithfully emulates its target computer when running on a given host computer.
II. Validate that a particular rendering program runs correctly under emulation.
III. Validate that a particular record behaves under emulation as it did when rendered originally.

If (I) is true, then both of the following should logically be true:

- Any program that ran on the target computer should run identically under the emulator [97].

---

records. Similar standards may exist or may need to be created for other dimensions of fidelity, such as sound, movement, execution speed, etc.

[94] This may be warranted in extreme cases, such as the physical construction of Charles Babbage's 1837 design for the Analytic Engine, the first programmable computer.

[95] One such example was suggested above in the discussion of displaying an Excel 2000 spreadsheet on a year 2300 computer, where it was suggested that scrolling the Excel window might be accomplished by moving one's eyes and blinking rather than using a mouse.

[96] Some approaches advocate saving the original formats of records as well as transformed versions, but since they offer no way of interpreting the original format in the future, this will be of value only if some as-yet-unknown mechanism is developed for interpreting obsolete formats. If sufficient documentation is saved for each format (as is also advocated by some approaches), then future digital archaeologists may be able to use this to interpret obsolete formats on demand, but this encounters the recursive problem mentioned above (see note 49), wherein any such documentation that is saved in digital form must also be interpretable in the future.

[97] In a sense this is just a tautology, since this is precisely what it means to say that the emulator faithfully emulates its target computer. If some program that ran on the target computer refuses to run (or runs differently) under the emulator, then the emulator is not faithfully emulating the target computer.

- Any record that is rendered by any program that is run under the emulator should behave exactly as it did when rendered by that program on the record's original computer (i.e., the target computer).

These assertions beg the question, somewhat, since it may be difficult to prove that an emulator faithfully emulates its target computer [98]. Nevertheless, it should be possible to validate that the emulator correctly performs each of the target computer's operations in the order specified by any program. This is just the logical definition of what any computer does: it executes well-defined operations in well-defined orders. If a computer were merely a logical device with no connection to the outside world, this would be enough. The logical operations of any computer can be specified in formal mathematical terms, which can in turn be proven valid.

But I/O complicates the situation, since some of the operations that a computer performs affect the outside world, such as producing visible pictures, sounds, or other effects, which may not be easily defined in formal mathematical terms. In order to validate that an emulator faithfully emulates its target computer's I/O behavior, it may be necessary to use a combination of automated formal tests (e.g., comparing displayed images on a bit wise basis) and subjective tests. As suggested elsewhere[23], a form of double blind "Turing Test" might be used to validate I/O behavior in this way. This would consist of having users try out an emulated system and a range of original systems to see if they can reliably distinguish between them, without knowing in advance which is which. If the emulated system is effectively indistinguishable from one or more of the range of original systems in such a test, the emulator would be considered valid.

If (II) is true, then any record rendered by the given rendering program should behave as it did when rendered originally on its original computer. Again, all aspects of the rendering program's behavior other than its I/O can be formally validated. But when I/O behavior is taken into account(which is obviously a crucial aspect of a rendering program's behavior), we may again need to resort to subjective testing.

Condition (III) above is the condition we are really interested in, i.e., whether each record is rendered correctly under emulation. The reason for introducing (I) and (II) is to try to avoid having to validate the behavior of every individual record, which in practice will be infeasible due to the huge number of such records.

The best strategy for validating emulation would seem to be (I), since it has the greatest "leverage" of any of the alternatives. That is, if we can convincingly validate a given emulator, we can in principle skip (II) and (III), since they should follow automatically.

However, even if we accomplish (I), we must still validate that our emulator runs correctly on successive future host computers. If we use an Emulation Virtual Machine as a way of porting emulators to future platforms, then we must revise (I) slightly (producing Ia) and add a new condition (Ib):

> Ia) Validate that an emulator faithfully emulates its target computer when running on the EVM

[98] If an emulator of a given target computer is to be validated against its target machine, the emulator should obviously be written before the target machine becomes unusable. There is normally some time during which an old platform is still usable while a new platform is gaining popularity, so the emulator should be written and validated during this time.

Ib) Validate the implementation of the EVM on a given host computer

Just as the EVM replaces the need to port each emulator to each new computer by the simpler requirement that we port the EVM itself to each new computer, it also replaces the need to validate each emulator on each new computer by the simpler requirement that we validate the EVM itself on each new computer. Since there is only one EVM in any given epoch as opposed to many emulators, this should require less work.

In summary, validation is still an open issue for all digital preservation approaches. The emulation-based preservation approach requires further research into the possible use of automated formal tests and Turing Tests to aid validation. However, no other proposed preservation approach has an adequate answer to the validation question either--and because most approaches do not attempt to provide access to original records, validation may be even more problematic for them than it is for emulation, since they cannot compare preserved records against their original forms.

5.2.3 Vernacular extraction

As discussed in Section 4.5 above, one of the potential advantages of emulation is its ability to facilitate vernacular extraction of future surrogate copies of records directly from the originals. Further research is required to determine whether facilities can be built into an emulation environment to enable it to perform automatic or semi-automatic vernacular extraction in the future by utilizing capabilities of its host platforms. An alternative to providing built-in facilities of this kind would be to write explicit data extraction and retrieval programs that would run on the EVM, thereby allowing them to run indefinitely [99].

5.2.4 Emulating I/O devices

As noted above, emulating a computer platform involves more than simply emulating its central processing unit (CPU). It may also be necessary to emulate I/O devices and other peripherals in order to correctly interpret and render digital records. Although critics of emulation-based preservation argue that emulating such peripheral devices poses an insurmountable challenge, this is contradicted by the fact that millions of users run emulators of display terminals, printers, modems, and numerous low-level communications devices every day, without ever being aware that they are doing so.

Input/output in most modern computer systems is performed in one of two ways. One method, referred to as "memory-mapped I/O" allocates some part of the system's main memory as belonging to the I/O device in question. For example, display screens are typically mapped to" display memory" which is logically part of the main memory space of the computer. In order to display something on the screen, the computer simply stores appropriate data into this display memory, and a display controller device draws the resulting images on the screen. In an emulated version of a computer that uses memory-mapped I/O of this kind, original rendering software would store data into the display memory, and the emulator would detect this action and perform some function equivalent to that originally performed by the display controller to cause appropriate images to appear on the host computer's display. There are a number of ways that an emulator might perform this task, ranging from detailed emulation of the original display controller to simply doing something equivalent, regardless of the original display controller's internal logic. The key, however, is that

[99] The UVC data extraction approach described in Section 4.1 above suggests how this might be done. This is also in keeping with the strategy advocated by the CAMiLEON project (see note 58).

the rendering software need not be modified in any way for this to work: it simply performs output the way it always did, by storing data into display memory, and the emulator translates this action into some result on the host computer's display, just as the original display controller produced a corresponding result on the original computer's display.

The second main I/O mechanism involves the use of some kind of port (or "channel") [100]. Typically, some special CPU instruction is performed to issue a command to the port in question, after which a stream of bits is either sent to or received from the port. Although this has performance characteristics that are very different from those of memory-mapped I/O, the two approaches are abstractly quite similar. In both cases, rendering software performs some action which results in a bitstream being sent to or received from some device; in both cases, an emulator must detect this action, intercept the associated bitstream, and cause the host system to do whatever is necessary to perform the desired I/O.

5.2.5 Emulating storage device access

In addition to interacting with their users (via their user interface), most rendering programs interact with various storage devices or media on the computers on which they run. A typical application program may perform such interaction indirectly by calling a function provided by the operating system (for example, to read or write a named file from a named disk drive), but some programs may bypass the operating system and perform such functions themselves. In any case, either the application itself or the OS will ultimately generate some kind of signal or command to address the desired device.

As in the case of emulating I/O devices, the preferred strategy here must be to avoid modifying the original software in any way, which implies that the emulator must intercept the signals or commands emanating from the emulated computer (via I/O ports, channels, busses or other such mechanisms) and respond to them appropriately. For example, suppose the original OS issues a command to read data over an I/O channel that was (in the original system) connected to a disk drive. This command will consist of some sequence of bits that the channel controller interprets as requesting a particular block of data from the disk. If this request is not already expressed in such terms, it is translated by the channel controller into a sequence of low-level commands to the disk drive itself, e.g., asking for a specified stream of bits from a specified set of sectors and tracks on the disk. Whether the request is stated in terms of named items or sectors and tracks, the emulator need only implement the appropriate protocol and do whatever it is asked to do (i.e., return the requested data from wherever it is stored on the host computer). That is, the emulator intercepts each request generated by the original software and does whatever it requests--using the host system's native storage media to achieve the desired result. If an original rendering program (or the OS that it invokes) refers to information on a storage device in terms of the structure of the original storage device, then the emulator must translate this into a corresponding storage request on the host system which will access the desired data.

Now suppose that in order to perform its task, some original rendering program used data files in particular formats stored on tape or disk drives on some now obsolete

[100] Note that this use of the term "port" is quite different from its meaning in the rest of this paper. Here, a port is an access point or doorway, analogous to a porthole in a ship, whereas "porting" a program from one computer to another means moving it from one place to another (as in the word "portable"). Familiar types of I/O port include serial, parallel, USB, Firewire, etc., some of which may be provided by "busses" such as USB, Firewire, SCSI, etc. For the purposes of this paper, it is unnecessary to distinguish between ports, busses and channels.

computer. Then saving this rendering program would have required saving those data files as well. In accordance with the emulation philosophy of not changing original bitstreams, saving these original data files will involve transferring their bitstreams verbatim to future storage media as necessary, retaining their original structure. That is, any original block, sector, track, or other addressing structure will be retained in the preserved bitstream (this is implied by saying that the bitstream will not be altered in any way). When the original rendering program runs under emulation in the future and issues a request for the data at a particular location in one of these data files, the emulator can use this original location information to access the saved data on the host computer. To do this, the emulator may have to translate the original location information into the corresponding location on the modern host computer's storage medium, but this does not require rewriting any of the original rendering software.

5.2.6 Networked and multi-machine systems

For simplicity, the discussion so far has centered around digital records that consist of single files rendered by single programs running on single computers. As noted above, more complex cases exist, in which records may consist of multiple files, databases, or data streams distributed across client-server or multi-tiered architectures running on multiple computers over networks. Interaction between multiple processes or tasks running on a given computer should pose no problem for emulation, since running the original operating system on an emulated computer should recreate the original task and process environment, including any inter-process communication and synchronization mechanisms that were available on the original computer. Systems relying on multiple computers would similarly be handled by emulating each of the computers in question, thereby allowing the original distributed software to work just as it did originally, assuming that any network or other connections between the original machines is also emulated.

In order to retain the central feature of the emulation approach, which is that no original software is ever rewritten, it is attractive to emulate all of the relevant computers and network hardware in such a configuration; however, this hardware can be emulated at a relatively high level. For example, the details of the network can be abstracted and emulated by a simple program that implements the network protocols without emulating the detailed network topology or the servers, routers, switches, or other hardware that implemented the original network. In order to emulate the performance of the original system, this network emulation may incorporate delays that are representative of the original network (based on performance statistics for the original network, recorded traffic histories, etc.), but this need not involve detailed emulation of the network itself [101]. Additional research is required to determine the proper level for emulation of such network and inter-machine protocols for preservation purposes.

5.2.7 Emulating multiple hardware and software configurations

A computer system consists of more than just a processor. Various auxiliary and peripheral devices(such as co-processors, graphics or sound chips or boards, memory, secondary storage, and input/output devices) may also be relevant to the behavior of a given program. It is therefore necessary to emulate entire hardware systems (or platforms) whose configurations include all relevant devices. There are at least two alternative approaches to achieving this. The simpler approach would emulate an entire configuration as a monolithic whole. Under this approach, each

[101] In fact, this would seem to be the only approach that would recreate the network's behavior. Actually emulating the network in detail would not be an advantage in this regard, since the original live traffic would no longer be available to cause the emulated network to reproduce the original network's behavior.

alternative configuration would require a separate emulation, but it should be possible to define maximally robust configurations, each of which combines as many optional components as feasible, to minimize the number of distinct configurations that must be emulated. For example, a maximal configuration might include the maximum amount of memory that can be installed in a given platform, the maximum number of different kinds of disk drives, and all compatible optional components, such as co-processors, sound boards, graphics accelerator cards, peripherals, etc.[102]

A more efficient and more flexible (but also more challenging) alternative would create modular emulators of individual hardware components that would be automatically (though virtually) "plugged together" to create different configurations, just as the original hardware components were physically plugged together. This would allow an original, specific hardware configuration to be recreated by the emulator [103].

Similarly, different types of digital records – or even different individual records--may require different combinations of software applications, utilities, plug-ins, drivers, library software, etc. Each such combination constitutes a different software configuration. There are at least two alternative approaches to solving this software configuration problem as well. It is possible (and perhaps safest) to preserve each such configuration as a separate bitstream, representing a "memory image" of all of the software that must be loaded in order to render a given record or type of record. To do this, the original software would be loaded in its executable state on its original platform, and the resulting contents of the memory and/or disk drive of that platform would be saved [104].
A more efficient (but also more challenging) alternative would be to preserve the bitstream of each software component separately and automatically "load" these bit streams onto the emulated platform to create the required software configuration, just as the original software was loaded onto its original platform. Running the original OS on an emulated system should perform much of this loading process automatically.

Although configuration problems can be troublesome, the vast majority of programs run on their intended host platforms without encountering such problems; if this were not the case, computers would be far less useful than they are. Nevertheless, both hardware and software configuration aspects of emulation require further exploration.

5.2.8 Metadata

The development of metadata is one area of digital preservation that has received significant research [105].

From one perspective, emulation can be viewed as a way of avoiding the need for most technical metadata. If emulation is employed, the hardware platform on which a digital record originally ran and any software it required are captured directly by the emulation approach itself, so there is no need to create and store metadata describing

[102] It is important to note that this is exactly how most software is currently run, i.e., on standardized platforms. If every program required a unique hardware configuration, we would never get any work done. The National Library of the Netherlands (Koninklijke Bibliotheek) refers to this as a "reference platform" [29].
[103] This is the approach taken by the SimOS [8, 16] and PROTEUS [2] efforts.
[104] Note that this is the initial approach identified by the Koninklijke Bibliotheek for their digital repository.
[105] Notable among such efforts are the Dublin Core [4], the work by Cox, et al., at the University of Pittsburgh [3], and the OAIS project [25]. However, most of these efforts focus on descriptive metadata needed to characterize the provenance and other intellectual aspects of a work; only a few publications [18, 22, 23, 25, 9] address the question of what specific technical metadata may be needed to facilitate the preservation of digital records, whether by means of emulation or any other approach.

them [106]. Nevertheless, metadata may be required to describe the procedure needed to recreate a record by means of emulation. It remains a question for future research what kinds of metadata are required by emulation-based preservation.

5.2.9 Intellectual property issues

The fundamental idea of using emulation for preservation is to allow running original rendering software. Such software is often proprietary and typically relies on additional proprietary software such as an operating system. Running such programs under emulation in the future therefore involves licensing issues. In addition, there are questions about the ethics and legality of writing emulator programs, which reproduce the behavior of hardware that may be patented or otherwise protected under intellectual property laws [107]. Issues such as these will have to be addressed at the policy level in order to create a sound legal basis for using emulation for preservation.

5.2.10 Process and cost models

As the emulation approach to preserving digital records is further researched and explored, appropriate process and cost models for the approach must be developed. Tentative models of these kinds for the archives and libraries communities have previously been presented [18, 22, 23], but it may be premature to develop such models any further until additional experience has been gained with emulation-based preservation.

[106] In fact, it can be argued [23] that keeping metadata about the software or hardware on which a digital record originally relied is neither necessary nor sufficient for reproducing its behavior and can therefore be avoided except for historical purposes. Whereas it may be of historical interest to know what software and hardware a record used, it is not necessary to record this information as metadata if emulation is being used, because emulation implicitly captures the same information in executable form. Neither is it sufficient, because such metadata cannot recreate the record's original behavior unless the relevant software can be run on that hardware (which is precisely what emulation does, by virtually recreating the hardware and saving the actual software). Emulation provides a direct, operational means of linking a digital record to its required software and hardware, rather than merely saving their names or descriptions. The linkage that an emulation environment provides between a digital record and the software and emulated hardware it needs to run might be considered a kind of metadata, but this seems to distort the usual meaning of that term.

[107] Throughout the history of computer science, emulation has been accepted as a legal technique; however, some manufacturers of game platforms have challenged certain aspects of emulation in recent years. Of course, if emulators are provided by the vendors of the emulated hardware, this should not be an issue.

6 *Conclusion: the advantages of emulation*

Emulation is the only solution to digital preservation proposed so far that offers a way of preserving a digital record in its original form, i.e., as a digital-original. This appears to be the only way to ensure that the form and content of a record have not been corrupted--or even to test whether they have been. Furthermore, no other proposed solution offers a way of preserving the complete look and feel and the dynamic, active or interactive behavior of executable and inherently-digital records.

Emulation offers a conceptually simple and universal way of preserving artifacts in virtually all digital formats, with the marginal cost of preserving each additional format being merely the cost of saving its original rendering software. This universality and low incremental cost make emulation quite attractive as a "catchall" or default preservation approach, whether or not it is used as a primary approach. In addition to serving as a backup preservation mechanism for records that may be preserved by other means as well, emulation offers the ability to save such records in their digital-original forms, and it offers a way of saving records that utilize marginal or obscure formats, which may be too expensive to preserve by other means. Moreover, emulation requires virtually no processing of individual records and can be implemented merely by writing a single program to emulate each old computer as it becomes obsolete and another single program to host an Emulation Virtual Machine on each new generation of computer. Although the use of emulation to preserve digital records still requires resolving some significant issues, its potential low cost, universality, and ability to preserve originals--along with all of their inherently-digital aspects--argue that it is well worth pursuing.

7 *Bibliography*

1. Bearman, David, "Reality and Chimeras in the Preservation of Electronic Records", D-Lib Magazine, april1999 (http://www.dlib.org/dlib/april99/bearman/04bearman.html).

2. Brewer, E., A. Colbrook, C. Dellarocas, and W. Weihl, "PROTEUS: A high-performance parallel-architecture simulator," Performance Evaluation Review, deel 20, nr. 1, blz. 247-248, juni 1992, http://citeseer.nj.nec.com/brewer91proteus.html .

3. 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, nr. 4 (1994), blz. 279-300.

4. Dublin Core Metadata (http://purl.org/metadata/dublin_core)

5. Gilheany, S., "Preserving Information Forever and a Call for Emulators," gepresenteerd op Digital Libraries Asia 98: The Digital Era: Implications, Challenges & Issues, 17-20 maart 1998, Singapore http://www.archivebuilders.com/pdf/22010v052.pdf

6. Granger, Stewart, "Emulation as a Digital Preservation Strategy," D-Lib Magazine, oktober 2000, http://www.dlib.org/dlib/october00/granger/10granger.html .

7. Granger, Stewart, Digital Preservation & the CAMiLEON PROJECT, http://ds.dial.pipex.com/stewartg/cam-london.html.

8. Herrod, Stephen A., Using Complete Machine Simulation to Understand Computer System Behavior, Ph.D. Thesis, Stanford University, februari 1998 (http://www-flash.stanford.edu/~herrod/docs/thesis-2sided.pdf).

9. Holdsworth, David en Paul Wheatley, Emulation, Preservation and Abstraction, http://129.11.152.25/CAMiLEON/dh/ep5.html, 2001

10. Holdsworth, David, C –ing ahead for digital longevity, http://www.si.umich.edu/CAMILEON/reports/cingahd.html

11. Lorie, Raymond, Long-term Archiving of Digital Information, IBM Research Report, nr. RJ 10185, maart 2000, gereviseerd in juli 2000 (gereviseerde versie in de Proceedings of the First Joint Conference on Digital Libraries, Roanoke,VA, juni 2001).

12. Lorie, Raymond, "A Project on Preservation of Digital Data", RLG DigiNews, 15 juni 2001, deel 5, nummer 3, ISSN 1093-5371 (http://www.rlg.org/preserv/diginews/diginews5-3.html#feature2).

13. McKemmish, S. en D. Parer, "Towards Frameworks for Standardising Recordkeeping Metadata," Archives and Manuscripts, 26 (1) 1998, http://rcrg.dstc.edu.au/publications/recordkeepingmetadata/smckrmp1.html .

14. Mellor, P, Wheatley, P, Sergeant, D., "Migration on Request: A practical technique for digital preservation," ECDL 2002, http://www.si.umich.edu/CAMILEON/reports/mor

15. Michelson, A., en J. Rothenberg, "Scholarly Communication and Information Technology: Exploring the Impact of Changes in the Research Process on Archives," American Archivist, deel 55, nr. 2 (lente), 1992.

16. Rosenblum, Mendel, Edouard Bugnion, Scott Devine en Steve Herrod, "Using the SimOS Machine Simulator to Study Complex Computer Systems," in ACM TOMACS Special Issue on Computer Simulation, 1997

17. Rothenberg, J., "The Nature of Modeling", in Artificial Intelligence, Simulation, and Modeling, L. Widman, K.Loparo, en N. Nielsen (bewerking), John Wiley & Sons, Inc., augustus 1989, blz. 75-92. (Herdrukt als N-3027-D ARPA, The RAND Corporation, november 1989.)

18. Rothenberg, J. en T. Bikson, Digital Preservation: Carrying Authentic, Understandable and Usable Digital Records Through Time, Report to the Dutch National Archives and Ministry of the Interior by RAND-Europe, Den Haag, 1999 (beschikbaar als http://www.digitaleduurzaamheid.nl/bibliotheek/docs/final-report_4.pdf)

19. Rothenberg, J., Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation A Report to the Council on Library & Information Resources (CLIR), januari 1999, ISBN 1-887334-63-7 (beschikbaar als http://www.clir.org/pubs/reports/rothenberg/contents.html of http://www.clir.org/pubs/reports/rothenberg/pub77.pdf

20. Rothenberg, J., "Ensuring the Longevity of Digital Documents," Scientific American, deel 272, nr. 1, januari 1995, blz. 24-29.

21. Rothenberg, J., "Preserving Authentic Digital Information," in Authenticity in a Digital Environment, the Council on Library & Information Resources (CLIR), mei 2000, ISBN 1-887334-63-7, blz. 51-68, http://www.clir.org/pubs/reports/pub92/pub92.pdf.

22. Rothenberg, J., Using Emulation to Preserve Digital Documents, Koninklijke Bibliotheek, juli 2000, ISBN906259145-0, http://www.konbib.nl/kb/pr/fonds/emulation/emulation-en.html ; http://www.konbib.nl/kb/pr/fonds/emulation/usingemulation.pdf

23. Rothenberg, J., An Experiment in Using Emulation to Preserve Digital Publications, Koninklijke Bibliotheek, april 2000, ISBN 9062 59 1442, http://www.konbib.nl/coop/nedlib/results/emulationpreservationreport.pdf

24. Rothenberg, J., "Emulation for the Long-term Preservation of Digital Artifacts", The Information Management Journal, maart/april 2002, ISSN 1535-2897, ARMA International, Prairie Village, KS (http://www.arma.org).

25. Sawyer, D. en L. Reich, The Open Archival Information System (OAIS) Reference Model, White Book, uitgave 5, CCSDS 650.0-W-5.0 (http://ssdoo.gsfc.nasa.gov/nost/isoas/ref_model.html).

26. Swade, Doron, "Preserving Software in an Object-Centred Culture," in History and Electronic Artefacts, Edward Higgs, bewerking, Oxford: Clarendon Press, 1998, blz. 195-206.

27. Swade, Doran, "Virtual Objects – Threat or Salvation," in Museums of Modern Science (1999, NS 112), Tore Frangsmyr, Svante Lindqvist, mei 28-30, The Royal Swedish Academy of Sciences, Stockholm, Proceedings: Lindqvist (bewerking); Museums of Modern Science, Science History Publications, (2000), VS. ISBN: 0-88135-299-3;ISSN: 1404-7586, blz. 139-47.

28. Thibodeau, K., "Overview of Technological Approaches to Digital Preservation and Challenges in Coming Years," The State of Digital Preservation: An International Perspective, Council on Library and Information Resources (CLIR), 1755 Massachusetts Avenue, NW, Suite 500, Washington, DC 20036, ISBN 1-887334-92-0; http://www.clir.org/pubs/abstract/pub107abst.html

29. van der Werf, Titia, "Experience of the National Library of the Netherlands," in The State of Digital Preservation: An International Perspective, Documentation Abstracts, Inc., Institutes for Information Science, Washington, D.C., 24-25 april 2002, Conference Proceedings, Council on Library and Information Resources (CLIR), Washington, D.C. juli 2002, http://www.clir.org/pubs/abstract/pub107abst.html, pp. 54-64.