An Experiment in Using Emulation
to
Preserve Digital Publications

Jeff Rothenberg
RAND-Europe

April 2000

Published by
The Koninklijke Bibliotheek
Den Haag

Revision 2000-05-11

Title: An Experiment in Using Emulation to Preserve Digital Publications

ISBN (applied for)

Author: Jeff Rothenberg

Date: April 2000

This study was commissioned by the Koninklijke Bibliotheek, The National Library of the Netherlands, and jointly funded by the European Commission’s Telematics for Libraries Programme.

PREFACE

This report presents the results of a small study undertaken by RAND-Europe for the National Library of the Netherlands (the Koninklijke Bibliotheek or "KB") in connection with their work on the NEDLIB (Networked European Deposit Library) effort (described below), jointly funded by the European Commission's Telematics for Libraries Programme. This study was performed as the first phase of a longer-term effort by the KB to test and evaluate the feasibility of using emulation as a means of preserving digital publications in accessible, authentic, and usable form within a deposit library.

Since this report is intended to document the results of this study for the KB, it relies heavily on the KB's understanding of the context in which the study was performed, including issues concerning the background of the problem, the motivation for the study, and the choice of approach and methodology. Nevertheless, it is hoped that the discussion may be of interest to other members of the library community as well as archivists, government recordkeepers, and preservationists concerned with the problem of preserving information in the digital age.

SUMMARY

This report presents the results of a small study undertaken by RAND-Europe for the National Library of the Netherlands (the Koninklijke Bibliotheek, or “KB”) in connection with their work on the NEDLIB (Networked European Deposit Library) effort (described below), jointly funded by the European Commission’s Telematics for Libraries Programme. This study was intended to be the first phase of a longer-term effort by the KB to test and evaluate the feasibility of using emulation as a means of preserving digital publications in accessible, authentic, and usable form within a deposit library.

The increasing use of digital technology to produce documents, databases, and publications has a serious flaw: there are so far no available techniques for ensuring that digital information will remain accessible, readable, and usable in the future. Unless libraries, archives, government agencies, and other recordkeeping organizations find ways to ensure the longevity of digital artifacts, considerable amounts of valuable information may be lost forever.

Within the library and archives communities, it is now generally recognized that digital information must be copied to new storage media quite frequently, since such media become physically unreadable or obsolete within a few short years. But there is a more complicated impediment to preserving the digital information stored on such media: Digital information can be rendered usable only by running appropriate software, and such software—along with the hardware on which it runs—becomes obsolete at least as quickly as the media on which the information is stored. The software that must be run to make a given digital document intelligible and usable must correctly interpret the “logical format” of the document. Without such appropriate software, trying to read a digital document is like trying to read hieroglyphics without a Rosetta Stone.

The approach most often proposed for digital preservation involves “migration” (i.e., conversion) of a document from its original logical format into successive subsequent formats as each previous format becomes obsolete. Yet in addition to being highly labor-intensive (since every document must be converted in this way every time its current logical format becomes obsolete) such conversion is quite likely to corrupt a document, destroying its original appearance, structure, interactive behavior, its “look-and-feel” and even its actual content. Moreover, since each conversion is performed on the result of the previous conversion, such corruption is cumulative—and the original document cannot be used to correct or even detect such corruption, since the original becomes unusable soon after it is first converted.

The only way to avoid this kind of corruption is to use a digital document’s original application software to interpret its original logical format; but this requires somehow keeping that original software executable indefinitely. Emulation (in particular, using software to allow future computers to reproduce the behavior of obsolete computers) appears to be one way of making such original software executable far

into the future. Emulation is a widely used technique, in which one computer system reproduces the behavior of another system. Though it has not yet been applied to preserving digital documents in any systematic way, neither has any other approach.

The fundamental idea of the work described here was to test whether emulating obsolete computer hardware on future computer hardware can be used to confer longevity on digital publications by allowing their obsolete software to be run on future platforms. This report describes the first phase of this work, which involved developing a prototype experimental environment for trying out emulation-based preservation, using commercial off-the-shelf emulation tools to produce an initial proof-of-concept. The experimental conditions and environment are described along with results and observations arising from the initial experiments that were conducted. Results include proposed data, metadata, and procedural models to support emulation-based preservation, as well as recommendations for future experiments and experimental procedures. The analysis is presented in the context of the increasingly accepted Open Archival Information System (OAIS) as well as the NEDLIB adaptation of the OAIS, the Deposit System for Electronic Publications (DSEP), both of which are discussed below.

As a result of the initial 1999 experiment, a specific, concrete proposal is presented for implementing emulation-based preservation: the various steps necessary to preserve digital publications by means of emulation are described and associated with the appropriate processes in the DSEP model.

Although the 1999 experiment was conducted somewhat informally (its main purpose being to develop procedures for future iterations) its results nevertheless confirm that the overall experimental framework is feasible. Moreover, even using off-the-shelf emulation (which was not designed to be used for preservation purposes) the experimental outcome was striking: The appearance and behavior of digital publications under emulation was found to be virtually indistinguishable from their appearance and behavior on their original platforms. While this is far from conclusive (since many further questions remain to be answered) it is an indication that emulation-based preservation has significant potential and warrants further exploration.

TABLE OF CONTENTS

FIGURES

1. Objectives

This report presents the results of a short study undertaken by RAND-Europe for the Koninklijke Bibliotheek (the National Library of the Netherlands, hereafter referred to as the "KB") in connection with their work on the NEDLIB (Networked European Deposit Library) effort,[1] jointly funded by the European Commission's Telematics for Libraries Programme.

The study described here was intended as the first phase (referred to herein as the "1999 iteration") of a multi-phase effort by the KB, extending beyond the immediate needs of the NEDLIB project. This long-term effort is intended to test and evaluate the feasibility of using emulation as a means of preserving digital publications in accessible, authentic, and usable form within a deposit library—and to incrementally implement an experimental strategy based on this approach within an evolving digital repository.[2]

This multi-phase effort is envisioned as a spiral process, i.e., one in which the exact tasks to be performed in each iteration are to be decided on the basis of the outcome of previous iterations (or "spirals").[3] In general, each iteration will conduct an experiment to test the use of emulation as an approach to preserving digital publications.[4] The fundamental purpose of these experiments is to test whether emulating obsolete computer hardware on future computer hardware can be used to confer longevity on digital publications by allowing their obsolete software to be run on future platforms.[5]

---

1 NEDLIB is a collaborative project of a number of European national libraries, which aims to construct the basic infrastructure upon which a networked European deposit library can be built. The objectives of NEDLIB concur with the mission of national deposit libraries to ensure that electronic publications of the present can be used now and in the future. See http://www.konbib.nl/coop/nedlib/index.html for further information.

2 For discussions of the problem of digital preservation as well as the motivation and potential for using hardware emulation as a preservation approach, see Jeff Rothenberg, "Ensuring the Longevity of Digital Documents", *Scientific American*, January 1995 (Vol. 272, Number 1), pp. 42-7; Jeff Rothenberg, *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation, A Report to the Council on Library & Information Resources (CLIR)*, January 1999, ISBN 1-887334-63-7; Jeff Rothenberg and Tora 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, The Hague, 1999.

3 The spiral development model was introduced by Barry Boehm (see Boehm, B. W., "A Spiral Model of Software Development and Enhancement," *Computer*, Vol. 21, No. 5, May 1988, pp. 61-72). This approach has been widely adopted, particularly in projects involving the development of experimental techniques. It is well suited to such projects because it focuses each spiral on the most serious apparent risks (or unsolved problems) facing the project as it unfolds.

4 Most of the issues and approaches discussed herein apply to the full range of digital data, documents, records, and related artifacts, whether or not they are considered "publications" (as defined by a library of deposit). For simplicity, the terms "document" and "publication" will be used throughout this report to include all such artifacts.

5 Note that it may be necessary for the specifications (the instruction set, etc.) of obsolete systems to be available in the public domain (or at least under license) for this to be viable.

In any particular experiment, at least two hardware configurations must be distinguished, namely, the original system that will be emulated and a different ("host") system on which the emulation will be performed. Early experiments may use existing "off-the-shelf" emulators as a proof of feasibility, whereas later experiments may develop prototype emulators; however, even the earliest experiments must attempt to describe the kind of compiler or other process that would be needed to produce an emulator of the required type, using the specifications of the original system as input.

Each experiment must select examples of digital publications along with the application software needed to access, render, and view them on their original platform. A plan must then be developed for testing and comparing the behavior of the sample digital publications on the original system and on the emulated system. The results of this behavioral comparison are to be evaluated according to "authenticity criteria" for the given types of publications, to be developed as part of each experiment; these criteria are intended to "validate" the behavior of publications under emulation.

Each experiment must also develop functional and data models representing the experimental procedure itself, all relevant data and metadata elements, the relations between these elements, the functionality required by the experiment, and the processing of digital publications that is performed within the experiment. These models are to be mapped to the Open Archival Information System (OAIS) reference model,[6] as extended in the NEDLIB Deposit System for Electronic Publications (DSEP) design.[7]

In cooperation with the Information and Communication Technology (ICT) department of the KB, each experiment is to be implemented and conducted within the test environment of the Depot Nederlandse Elektronische Publicaties (DNEP) system[8] in order to study the procedural and organizational aspects of the emulation approach.

The 1999 iteration of this work consisted of performing a first spiral of the experimental process described above, utilizing existing commercial emulation

---

6 The OAIS is a proposed and increasingly accepted reference model for describing digital archives and repositories. It proposes a standard terminology for digital archives as well as a conceptual framework for building digital repositories, and it discusses many issues surrounding the creation of such repositories. See *Reference Model for an Open Archival Information System (OAIS)*, CCSDS 650.0-R-1, ("Red Book"), May 1999, (available at http://ssdoo.gsfc.nasa.gov/nost/isoas/ref_model.html).

7 The DSEP is a proposed system to be developed to serve as an evolving digital repository for NEDLIB. Its design is generally based on the OAIS reference model, though it extends and modifies that model in a number of ways. See *Data/functional model for a DSEP*, version 0.1 Nedlib document GEN-264. http://www.konbib.nl/coop/nedlib/meetings/paris/GEN-264.pdf and T. van der Werf-Davelaar, "Long-term Preservation of Electronic Publications: The NEDLIB project," *D-Lib*, Vol 5, No. 9, Sept 1999 (ISSN 1082-9873; http://www.dlib.org/dlib/september99/09contents.html).

8 The DNEP is the digital publication repository that is to be implemented by the DSEP.

software to identify requirements for using emulation as a means of providing digital longevity in the deposit library context. The next two sections present the results of this initial iteration, while Sections 4 through 7 discuss the methodology that was employed in performing the experiment, as well as the details of the experimental environment and implications for future iterations.

2. A process model for using emulation to preserve digital publications

This section describes a proposed emulation process for preserving electronic publications, based on the author's ongoing research concerning this approach. Details of the emulation process are presented within the context of the current OAIS Reference Model. Section 2.1 discusses this reference model (and its adaptation by NEDLIB into the DSEP model) and the danger of interpreting any such model as a logical or functional design model. Section 2.2 then describes a specific approach to using emulation for digital preservation. Finally, Sections 2.3 - 2.7 show how this approach to emulation would affect the logical processes comprising the DSEP.

2.1 The OAIS reference model

The OAIS is a proposed "reference model" for archival preservation of digital information. It has been of great value in providing a comprehensive and consistent frame of reference that encompasses many of the issues surrounding the creation of digital repositories. The data flow model of the OAIS is shown in Figure 1. One of the

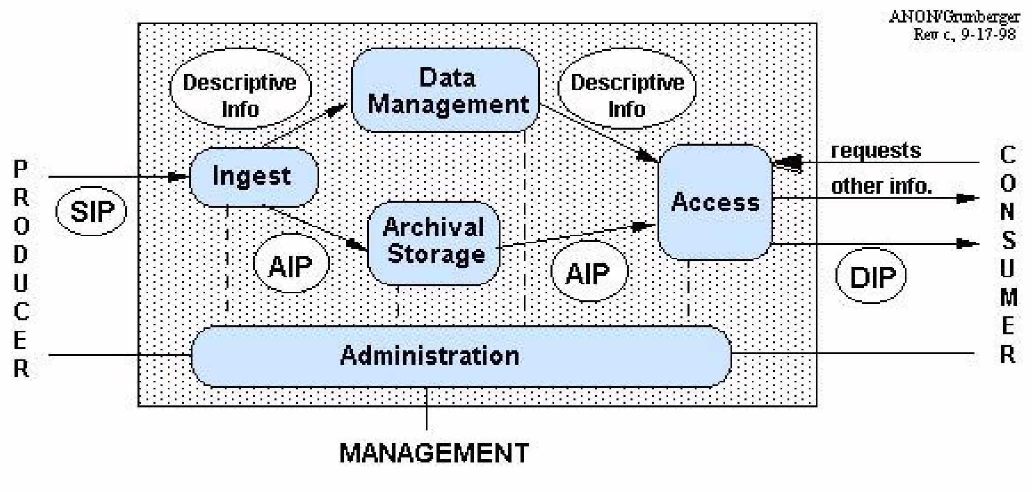


Figure 1: OAIS Data Flow Diagram

key concepts of the OAIS is that of an “Archival Information Package” (AIP), which consists of a specific digital artifact to be preserved (such as a document or record) along with whatever metadata and contextual information are needed to make it meaningful and usable. For example, an AIP might consist of a package containing a digital document along with metadata describing its provenance, its format, etc. An information item to be stored in the repository is submitted as a “Submission Information Package” (SIP), which the system augments, reformats, or repackages as necessary to turn it into an AIP. When users request information from the repository, an AIP is again repackaged and modified as necessary to produce a “Dissemination Information Package” (DIP), as shown in the diagram.

By offering a standardized terminology and viewpoint, the OAIS has facilitated coherent dialogue and cross-fertilization of ideas surrounding digital storage and access across such diverse communities as archives, libraries, and data warehouses.

Yet despite the ubiquitous use of the term “archival” in its documentation, the OAIS model is relatively silent on the subject of preservation per se: it focuses on the processes of describing, packaging and manipulating stored information with relatively little discussion of how to keep it meaningfully readable and usable in the future. It implicitly accepts the use of migration, i.e., transformation, of digital documents into new logical formats as their native formats—or the software or hardware on which they depend—become obsolete.[9] Section 4.2.1.4.1 of the OAIS contains the core of this discussion, introducing the notion of Representation Information, which may in turn require additional Representation Information to be intelligible, resulting in a recursive Representation Network.[10] The OAIS points out that “The Representation Information can also be viewed as including software that supports the presentation of the Content Information to the Consumer” thereby terminating the recursion of a Representation Network; but it rightly argues that “the reliance on working software can provide major problems for Long-Term Preservation when that software ceases to function.” However, whereas emulation offers a potential solution to this problem, the OAIS dismisses emulation as “a major technical and economic risk” and assumes (without the support of any empirical

[9] Unfortunately, the OAIS uses the term “migration” to refer to a number of quite different processes, including the copying of bit streams onto new media; however, it also uses the term to mean format conversion (or “transformation”) which is the sense in which it is used here. Note that almost any conceivable digital preservation technique (certainly including both migration and emulation) will rely on preserving some (original or transformed) bit stream representing the preserved document in question. This will require “migration” in the first sense above, i.e., copying (or “refreshing”) bit streams onto new media as necessary. Nothing in this report should be taken to suggest that emulation-based preservation eliminates the need to preserve bit streams in this sense: in fact, emulation extends this requirement to the preservation of software and hardware-description bit streams as well as those of documents per se.

[10] This is discussed further in Section 3.3 below.

evidence) that migration is the most logical preservation approach, despite recognizing that:

> Digital migrations are time consuming, costly, and expose [an archival information system] to greatly increased probabilities of information loss.[11]

Because it focuses only superficially on the technical issues involved in making digital information readable and understandable in the far future and because it implicitly assumes that migration will be the only method used to preserve such information, the OAIS presents an "archival" model that ironically says very little about preservation. Nevertheless, the model coherently organizes much of the remaining (i.e., non-preservation oriented) activity that an organization must perform in order to manage digital holdings, which has led to its growing adoption or adaptation.

One such adaptation is the augmented DSEP version of the OAIS model, shown in Figure 2, which was developed as part of the KB's NEDLIB effort. Among other things, this adds an explicit "Preservation" process to the OAIS model.[12] However, whereas viewing preservation as a separate process is a reasonably appropriate model for the activities that would be required by a migration-based approach to preservation, it is less successful at representing a preservation approach based on emulation. The use of emulation to preserve digital documents involves a number of coordinated activities which, as discussed below, cannot be localized in a separate preservation process but must rather be distributed through the processes Ingest, Archival Storage, Data Management, Access, and Preservation, as they are conceived in the DSEP model. In the interest of brevity, the following descriptions do not constitute complete revisions of the processes described in the DSEP document but address only those aspects of the processes that would be affected by the use of emulation as a preservation strategy.

It is important to keep in mind that the OAIS is intended as a reference model rather than a system design model. One implication of this (both for the OAIS and for the DSEP specification, which is derived from it) is that the functions or processes shown in these models do not necessarily correspond directly to the functional modules of a system that would implement that model. The functional decomposition of a system into appropriate modules is a design issue, and various implementations may well lend themselves to functional decompositions that are quite different from the

[11] Op cit, note 6, 5.1.1; p. 5-2.

[12] Op cit, note 7. The Preservation process that the DSEP adds to the OAIS is retained in the scheme of this report, since it is still needed to represent activities that do not rightly belong in other processes, despite the fact that not all of the activities that are requried to perform emulation-based preservation can logically be grouped within this separate Preservation process. The fact that the DSEP model needed to add a Preservation process to the OAIS model suggests that the OAIS model as it stands does not adequately address issues of preservation—even under its own assumption that migration is the only viable preservation method. Nevertheless, it is not the intent of this report to denigrate the OAIS, which has done a great service by providing a thoughtful and coherent conceptual framework for discussing many aspects of archival systems.

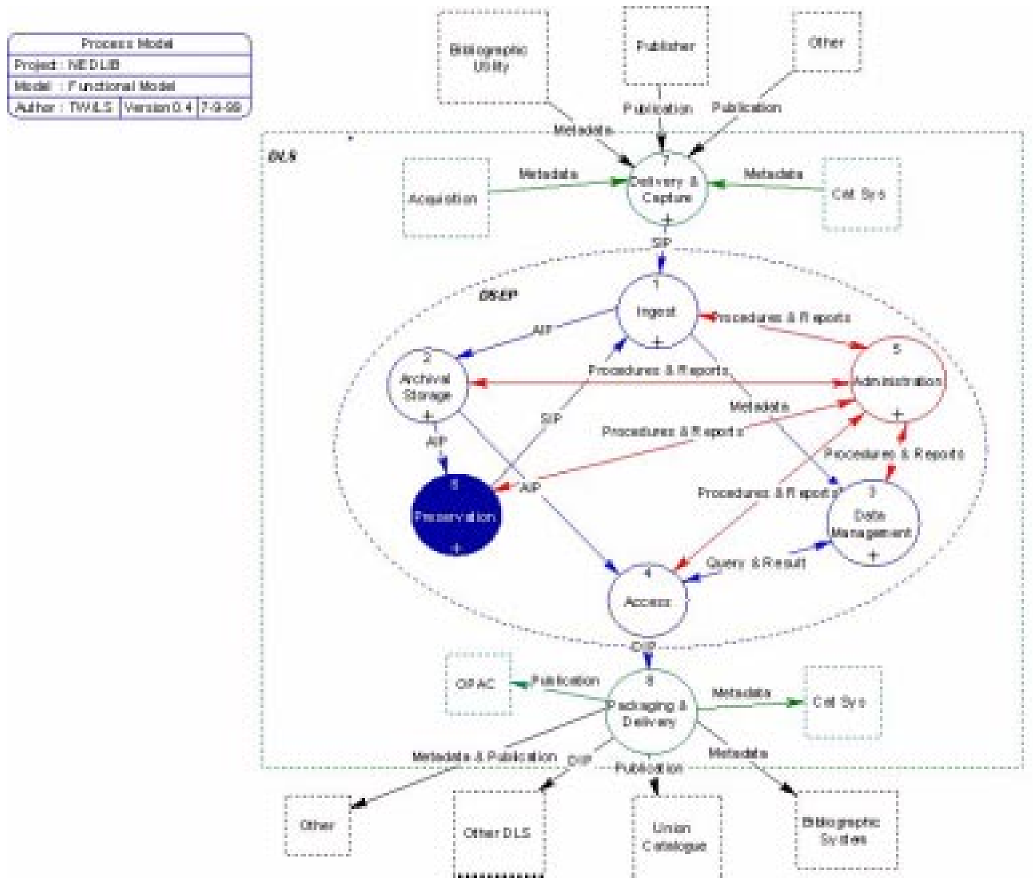


Figure 2: The DSEP process model

"reference processes" of the OAIS. Far from being a shortcoming, the reference orientation of the OAIS is a great strength, since it frees it (at least to an extent) from being bound to specific implementations. As pointed out above, even as a reference model, the OAIS is strongly oriented toward migration-based preservation as opposed to emulation, but it generally avoids implementation-dependence, making it more valuable by being applicable to a wider range of possible implementations. However, there is a danger that if the OAIS reference model is interpreted as a functional decomposition, the design and implementation of a system such as the DSEP may be detrimentally constrained.

Nevertheless, in order to develop emulation-based preservation beyond the concept stage, it is necessary to go beyond the conceptual "reference model" level and discuss issues that shade over into the design of the DSEP system, if not its implementation. For this reason, the following discussion will treat the processes shown in the draft DSEP specification as if they represented the actual logical processes to be performed within the DSEP, rather than as purely conceptual entities. This is still more abstract than interpreting them as functional modules in an actual implementation but appears to be consistent with the wording of the DSEP specification as well as with the apparent thinking of the KB.

Note that a similar issue surrounds the Archival Information Package (AIP) and related concepts in the OAIS and DSEP. The AIP concept can be interpreted as either a conceptual package or an actual data structure. If it is interpreted as a conceptual package, then a system designer is free to implement it using various different data structures; on the other hand, if the AIP is interpreted as the specification for an actual data structure, this design freedom disappears. For example, consider where various kinds of metadata should logically reside: while the OAIS implies that most or all metadata should reside within the AIP itself, this is presumably intended merely as a conceptual model for any actual design. A specific implementation of the AIP may therefore choose to factor this set of metadata elements into various subsets, each of which may logically "belong to" a different logical process and may physically reside in a different data structure. The actual implementation of an AIP (e.g., whether it should consist of one or more data structures in some programming language, fields in a database, files in a file system, etc.) should not be considered to be constrained by the OAIS.

As an example of exercising this freedom, the DSEP separates from the AIP all metadata elements except those that are considered part of the original publication. This allows the Data Management process to "own" and control all other metadata elements (such as those describing a document's current format, condition, location, usage, etc.). In particular, such information can be updated on a frequent basis and migrated to new data management software or representations as necessary, without having to modify the preserved publication itself, which remains inviolate in its AIP.

2.2 A concrete approach to emulation-based preservation

There are a number of alternative ways of implementing an emulation-based preservation approach, each of which may require somewhat different activities to be performed and somewhat different metadata to be kept. For the sake of concreteness, the following suggests one particular implementation, in which emulator specifications are expressed as programs in one of an evolving sequence of standard emulator specification languages; each such language is interpreted by an emulator specification interpreter program written for one of an evolving sequence of standard emulation virtual machines, which are designed to be easily hosted on future computing platforms.[13] A conceptual view of the relationships among these elements (suggesting how they would play together to perform preservation via emulation) is shown in Figure 3. A fragment of an actual hardware specification (for a Motorola 68000 CPU, expressed in Archelon's Retargetable Toolset format, coded in the C programming language), which could serve as an emulator specification, is shown in Figure 4.

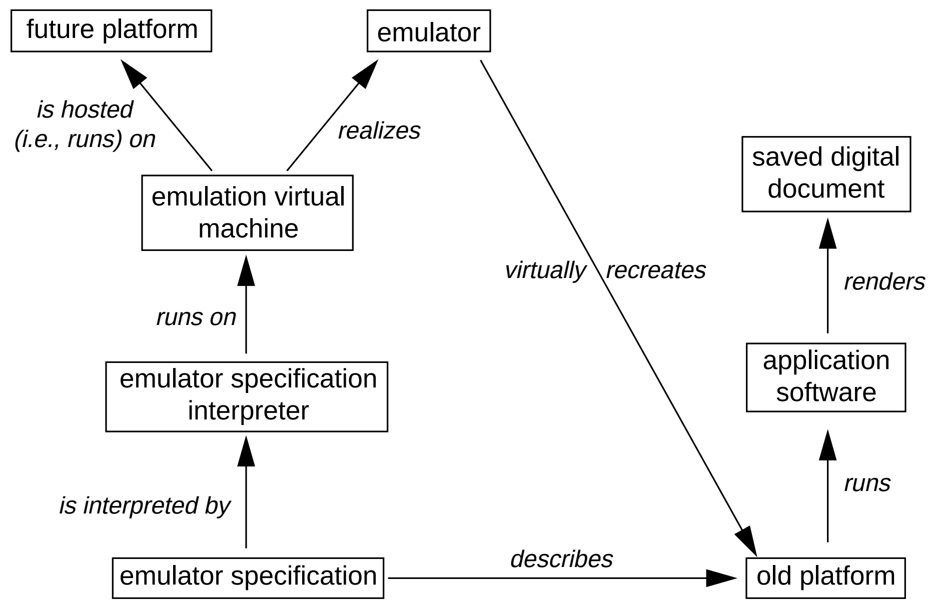


Figure 3: Elements of emulation-based preservation

This approach has the potential for minimizing the amount of work that must be done to use emulation for preservation. Emulator specification languages can evolve over time, but no emulator specification (for an obsolete computing platform) need ever be rewritten or translated, since it will always be possible to run its associated interpreter program (on successive emulation virtual machines). Similarly, although each new emulator specification language will require a new interpreter program to be written for it, no existing interpreter program need ever be rewritten or translated, since it will always be possible to run it on its intended emulation virtual machine. Finally, although emulation virtual machines may themselves evolve over time, older ones need never be rewritten or translated so long as a single emulator specification is written that allows each such virtual machine to be emulated by its successor.

[13] A virtual machine is something that serves the role of some computer that does not actually exist. Virtual machines are typically generalized or simplified analogues of real computers, implemented by software. Since a virtual machine serves the role of a computer, programs can be written to run on it, even though it itself is implemented as a program. Since a virtual machine is implemented by a program, that program must run on some "host" computer, but the host computer that it runs on is not (normally) the machine that is virtually created. A number of approaches are available for minimizing the effort needed to host a virtual machine on a new host platform, such as minimizing the number of "primitive" operations that must be implemented for it to run or running it on top of a simpler, lower-level virtual machine. Note that the only programs that emulation virtual machines must run are emulator specification interpreters.

Considerable care has been taken to ensure that the key elements of this approach (specifically, emulator specification languages and emulation virtual machines) are free to evolve over time, since no standard can be assumed to be eternal. It is hoped that the designs of such specification languages and virtual machines will be

```
# R E G I S T E R   S E T S
#
mau := 8; /* byte addressable memory */
 # 32 general purpose registers which can be used for any type and for any operand

regset := R[32] width=32
     optype=int,ptr,ptr2,float,double,longdbl,codeptr
     regtype=char,short,int,ptr,ptr2,codeptr,long,float,
         double,longdbl;
stkptr  := R[31];

# O P E R A N D S
#
# this section describes and names the various operands which
# you can use in assembly language instructions.
#
operand code_addr codeptr; # pointer to code memory
operand data_addr ptr;     # pointer to data memory
operand const16   sconst -32768 32767; # 16 bit signed constant
operand gp_reg    reg   R;  # general purpose register

    # an "amode" is an address mode. This is a memory reference.
    # the "ri_addr" amode forms an address by summing a register and a constant.
    # the "dir_addr" amode forms an address by direct address.

operand ri_addr   amode R+const16 format "%B,%O";
operand dir_addr  amode data_addr format "%O";

# F O R M A T S
#
# this section describes and names the various instruction formats supported by the hardware.
# You define a format as using zero or more operands.
#   "src" means the only source operand; "lsrc" means the left source operand
#   "rsrc" means the right source operand; "dest" means the destination operand

format mem_load_ri   src ri_addr  dest gp_reg;
format mem_load_dir  src dir_addr dest gp_reg;
format mem_store_rri src gp_reg   dest ri_addr;

# O P C O D E S
#
opcode ldri  mem_load_ri;
opcode ld    mem_load_dir;
opcode jmp  : jump branch;
```

Figure 4: Fragment of an emulator specification

relatively stable, having lifetimes on the order of ten years or more, which would make most of the actions described here (and in later sections) relatively rare events. However, nothing in this approach relies on the longevity of these elements, which can be expected to change many times over the course of the future.

The total inventory of products that must be created to implement this scheme (excluding documentation, which is discussed below) consists of the following:

- One emulator specification must be written (in the current emulator specification language) for each new family of platforms whose documents are to be preserved.[14]
- One program must be written to host the current emulation virtual machine on each new family of platforms that are to be used to access old, preserved documents.
- One emulator specification interpreter program must be written (which runs under the current emulation virtual machine) to interpret each new emulator specification language that is defined.
- If and when a new emulation virtual machine design is deemed necessary
  - — One emulator specification interpreter program must be written that runs on the new emulation virtual machine and interprets the current (possibly new) emulator specification language; and
  - — One emulator specification must be written—in the current (possibly new) emulator specification language—that describes the old emulation virtual machine that is being replaced.[15]

Under this implementation of the emulation approach, rendering a saved digital document in the future requires that:

a) The bit stream of the original document is preserved;

b) The bit streams of all software needed to render the document are preserved;

c) Emulator specifications—written in one of the accepted emulator specification languages—for all of the hardware needed to run the document's software are preserved (minimally as bit streams, though they may also be preserved in human-readable form for redundancy[16]);

[14] A "family of platforms whose documents are to be preserved" is one for which some document that is to be preserved requires software that runs only (or preferably) on some platform in that family.

[15] This allows all old emulator specification interpreter programs to run (under emulation) on the new emulation virtual machine, thereby allowing all old emulator specifications to be interpreted.

d) An emulator specification interpreter program—written to run on one of the accepted emulation virtual machines—for the emulator specification language used in (c) is preserved (minimally as a bit stream, though it may also be preserved in human-readable form for redundancy);

e) An emulator specification for each accepted emulation virtual machine is preserved (minimally as a bit stream) written in an emulator specification language for which there is an interpreter that runs on the next accepted emulation virtual machine in the sequence of such machines, thereby allowing each new emulation virtual machine to emulate its immediate predecessor;

f) Syntactic and semantic specifications for the current emulator specification language are maintained in human-readable form for as long as it remains the current standard, to facilitate writing emulator specifications for new hardware platforms;

g) Specifications for the current standard emulation virtual machine are maintained in human-readable form for as long as it remains the current standard, to facilitate both writing emulator specification interpreters to run on this virtual machine and porting the virtual machine itself to new hardware platforms.

Of these, only (f) and (g) must be preserved in human-readable form, and then only for the duration of a given emulator specification language or emulation virtual machine's respective tenure as the current standard. All of (a) - (e) may be represented as bit streams, whose preservation is relatively trivial.[17] Preserving (f) and (g) in human-readable form may require conversion, i.e., if the preferred human-readable documentation format changes during their tenure. However, this conversion should occur naturally, since both the emulator specification language and the emulation virtual machine specification will be used on an ongoing basis, the former to write emulator specifications for new hardware platforms, and the latter both when writing emulator specification interpreters to run on the current virtual machine and when porting this virtual machine to new platforms.[18]

[16] The phrase "human-readable" is used throughout this report to mean information that can be interpreted and understood by humans with a minimum of effort at a given point in time. This definition is intentionally imprecise: it is intended to include digital forms that utilize encodings and rely on software that are standard and ubiquitously available at the time in question, while excluding digital forms that rely on encodings or software that have become obscure or obsolete, even if they were once standard.

[17] If human-readable versions of any of these products are desired for redundancy or safety, these should be stored as digital documents in some simple standard form—such as that used to encode administrative metadata—in order to maximize the chances of their remaining human-readable in the future without the need to resort to emulation. However, the emulation-based preservation scheme does not rely on saving any such human-readable products, except for the highest-level explanatory documentation that tells future users what they must do to use the emulation scheme to access preserved documents.

In order to ensure that this emulation approach will properly preserve digital documents, the following ongoing tasks must be performed (in any reasonable order). Note that the first three tasks are analogous or identical to tasks that must be performed by any preservation approach, except to the extent that task-1 adds software (including emulator specifications) to the collection of bit streams that must be copied into the future and that task-3 adds emulator-related information to the collection of items whose names must be maintained over time. The DSEP process names shown in square brackets at the end of each of these tasks are the processes that should logically perform these tasks—as elaborated in the following sections. These tasks address ongoing preservation, for the moment ignoring Ingest and Access; in particular, these tasks assume that Ingest stores (as appropriate AIPs) the application software needed to render preserved digital documents and the hardware emulation specifications needed to run that software under emulation. The following sections show how this emulation approach would be integrated into the full range of DSEP processes, including Ingest and Access.

task-1: Ensure that the bit streams of digital documents (including all associated software and emulator specifications) are copied without corruption to new storage media as needed to keep them readable and accessible. (Note: this is required by any digital preservation scheme, at least for the bit streams of the documents themselves.) [Archival Storage]

task-2: Ensure that any required explanatory metadata associated with digital documents remains understandable. This may require converting at least the topmost level of such explanations (sufficient to explain how to read lower levels of explanation) into successive “explanation formats” as previous such formats become obsolete. Ideally, any such conversions should be reversible. (Note that this too is required by any digital preservation scheme.[19]) [Data Management]

task-3: Maintain the linkages between named software and emulator specification components and configurations, both in each document’s metadata and in the stored (AIP) forms of those entities in the DSEP (or in remote repositories), and maintain similar linkages between metadata references to named emulator specification languages, emulator specification interpreters, emulator specifications, and emulation virtual machine specifications and implementations. [Data Management]

[18] If it is desired to preserve (f) or (g) beyond their tenure as current standards, they should be stored as discussed in note 17. This might be done, for example, to support future validation of the behavior of emulator specifications or direct reimplementation of older emulators.

[19] Migration-based preservation, by continually converting stored documents into new formats, should keep them readable in their own right, thereby reducing the need for much explanatory metadata. However, some such metadata must be kept for any preservation scheme—if only to provide indexing and access information—and any such metadata will have to be converted.

task-4: Retain the bit streams of the emulator specification interpreters for all emulator specification languages that have been used to specify any of the emulators required by any of the documents in the DSEP's collection. Since these specification languages are not expected to change very often, there should not be a large number of such interpreters, so this should not be a significant burden. [Archival Storage]

task-5: Whenever a new emulator specification language is developed, ensure that a new emulator specification interpreter is developed for it, to run on whatever is the current emulation virtual machine, and preserve this emulator specification interpreter as an AIP in the DSEP.[20] [Preservation]

task-6: Whenever a new emulation virtual machine is developed, ensure that an emulator specification of the previous emulation virtual machine is developed (in the current emulator specification language), so that the new emulation virtual machine will be able to run all previous emulators, and preserve this emulator specification as an AIP in the DSEP.[20] [Preservation]

task-7: Maintain documentation in human-readable form for both the current emulator specification language (to facilitate writing emulator specifications for new hardware platforms) and the current emulation virtual machine (to facilitate both writing emulator specification interpreters to run on it and porting it to run on new hardware platforms). [Preservation]

task-8: Ensure that the current emulation virtual machine always runs on some computing platform that is currently available within the DSEP. This may require porting or funding the porting of this virtual machine to new computing platforms as they replace older ones. [Preservation]

task-9: Create and preserve in human-readable form explanations for how to use emulation to access preserved digital documents and for how to maintain the emulation infrastructure required for preservation. [Preservation]

This approach to emulation-based preservation of digital documents requires the modification or extension of a number of OAIS/DSEP processes, as discussed in the following sections.[21] Note that those processes that are not discussed here would remain essentially unchanged from their descriptions in the DSEP.

[20] This should be a relatively infrequent event.

[21] Whether these modifications and extensions are considered to be significant changes to the OAIS or simply elaboration of the reference model is of less importance than the fact that the steps they represent are not currently disucssed in the OAIS framework.

2.3 Ingest (and Delivery & Capture)

The activities discussed in this section may be distributed among the following subprocesses in ways that should be fairly obvious:[22]

- Create Submission Information Package (SIP) [7.2]
- Validate SIP [1.2.1]
- Decline [1.2.3]
- Quality Assurance [1.3]
- Generate Descriptive Information [1.5]
- Generate Archival Information Package (AIP) [1.6]

An AIP that is to be preserved using emulation must include:

1. Appropriate information concerning all application and system software (including specific versions) required to render the Digital Objects that comprise its Content (using OAIS terminology); and

2. Appropriate information concerning the hardware platform and configuration required to run that software sufficient to allow emulating that hardware on future platforms.

This information may consist of long-lived names, human-readable descriptions, or long-lived pointers to the relevant target software and hardware emulation facilities. Platform configuration information may refer to generic or specific versions or configurations of platforms with optional specifications of specialized peripheral hardware required by the software in question.

The most logical implementation for such information would seem to be to use standardized, unambiguous, long-lived identifiers for specific versions of software packages and platform configurations and then to store such software and the emulator specifications for such platform configurations as separate AIPs (for clarity, these will be referred to here as Software AIPs and Emulator Specification AIPs, although these are not materially different from one another—or from any other AIPs—consisting simply of suitably packaged bit streams). Alternatively, all necessary software and emulator specifications may be included within the same AIP as the document itself: while this would be highly redundant for software or emulators that are frequently used, it might be useful for cases in which either of

[22] Subprocess numbers (shown in brackets) are from the DSEP specification, version 0.1, 22-07-99.

these components is unique or when added robustness is desired (since an AIP that contained all of this would not rely on the presence of these components in separate AIPs or on linkages to these separate components in the DSEP or elsewhere).

If a digital document submitted in an SIP requires software or emulator specifications that are not already present in the DSEP (and are not included in the SIP itself), then these missing components should be acquired by Ingest prior to accepting the SIP. The submitter may submit such additional components as separate SIPs, or Ingest may locate these components in other repositories; but if this information is not supplied by the creator or publisher of a digital publication, it must be acquired or added by either Delivery & Capture or Ingest. Since the creator may be best qualified to supply this information, it would seem to make the most sense for Delivery & Capture to check for its presence and attempt to acquire it from the publisher if it is missing. If this information is supplied, it should be validated, presumably by Ingest, since such validation appears to be beyond the scope of Delivery & Capture. Note that this includes cases in which a submitted document relies on software that is not yet in common use (and therefore not yet preserved in the DSEP) or in which a document's required software runs on a new hardware platform for which no emulator specification has yet been preserved in the DSEP. The DSEP must acquire a suitably packaged, executable form of any new software (e.g., a "load image" including all necessary system and support software, loaded in a runnable state), and it must acquire an emulator specification of any new hardware platform, possibly after instigating or initiating its creation.[23]

It might be advisable for the DSEP to offer publishers a remotely accessible online "Validation Service" that would allow them to validate their SIPs themselves before submitting them. This might include facilities for obtaining (and including in the SIP) the necessary standard identifiers for the relevant software versions and platform configurations, and it might offer an emulation testing capability that would allow publishers to verify that their publications will be usable under emulation before they submit them. Making such a facility available to publishers could greatly reduce the

[23] This should be the preferred approach, since it allows verifying at the time of submission that a submitted document will be preservable by means of emulation. To allow such "up-front" verification, all of the components of the emulation scheme that are required to preserve a given document should exist within the DSEP by the time the document is packaged into an AIP. This implies that before the first document to use some new piece of application software, running under some new operating system on some new hardware platform is accepted by the DSEP, all required software and the emulator specifications for all required hardware should already be preserved in the DSEP. Among other things, this argues that whenever a new hardware platform is created, an emulator specification for that new platform should be created (and preserved in the DSEP) as soon as it becomes clear that the new platform is likely to be required by software that will be used to create documents that will need to be preserved. If an emulator specification for this platform does not exist when an SIP is submitted that requires it, Ingest would have to locate or instigate the creation of such a specification in order to support up-front verification, possibly postponing acceptance of the SIP until the specification was acquired. Waiting until a hardware platform is about to become obsolete before creating an emulator specification for it would require either postponing the acceptance of any submissions that depended on that platform until it is about to become obsolete or accepting them without already having all of the components required to preserve them, either of which would prohibit up-front verification.

validation effort required during Ingest while simultaneously ensuring that submitted publications will be properly preserved.

In some cases, Ingest might allow an SIP to contain an executable copy of some specialized software or an emulator description for some specialized hardware platform or component that is required by the publication being submitted.[24] If an included component of this sort is not already available in the DSEP (i.e., as a Software or Emulator Specification AIP) and it is packaged in such a way as to be easily extracted from the SIP, Ingest may choose to extract the software and save it as a separate AIP. In such cases, the completeness and suitability of the form of any included component must be validated by Ingest, and any packaging or representation conventions that are required by the DSEP must be enforced at this point, converting the information in the SIP, if necessary, to conform to these conventions. The component may then optionally be removed from the SIP and replaced by suitable naming information to allow accessing it as a separate AIP in the DSEP. If this is allowed, the form of the SIP should be designed to facilitate the extraction of components in this way. Alternatively, specialized software or emulator descriptions could be submitted as separate SIPs, thereby obviating the need for their extraction in Ingest.

Finally, Ingest should validate that the specified software (whether designated or provided) is executable by the specified emulator (whether designated or provided) and that it properly renders the Content Information in the SIP. While it may be impractical to validate every SIP in this much detail, this should at least be done for carefully chosen samples of submissions from given agencies or organizations.[25]

If some piece of software or emulator specification that is required by the publication submitted in a SIP is missing (i.e., is neither present in the DSEP already nor included in the SIP itself), the DSEP may reject the SIP (as in subprocess 1.2.3 "Decline"), or it may generate an attempt to acquire the missing component, either by finding it in some other existing repository or by instigating or initiating its development. The last of these options (initiating the development of a missing component) would presumably be exercised only for submissions of particular value or for missing components that are likely to be required by many submissions but are not expected to be supplied by other means. Nevertheless, since Ingest is logically positioned to detect such missing components, it should take responsibility (either as

[24] For example, CD-ROM publications, as currently packaged, consist of software bundled with the publication itself. Future publications might even include any required emulators in the bundle. While it is possible in such cases to generate a single AIP containing this entire bundle, thereby allowing the publication to execute its own software (and potentially its own emulator) from within its own AIP, this is inefficient if the bundled software or emulator are likely to be required by any other publication in the repository.

[25] As suggested above, in order to distribute the burden of performing such validation, the DSEP might provide a remotely accessible Validation Service that would allow submitters to validate their SIPs prior to submission; such a service would allow submitters to submit pre-validated SIPs to the DSEP, reducing the DSEP's validation role to that of verifying that pre-validation was in fact performed successfully.

part of subprocess 1.2.1 “Validate SIP” or as a new subprocess) for generating an alert whenever such missing components are detected. The DSEP administration can then decide whether to ignore the problem, search for the component elsewhere, motivate the submitter or some other organization to develop it, or initiate its development itself.

The use of standard forms for SIPs would greatly reduce the effort the DSEP would have to expend to validate submissions. Standards should be developed for software and hardware specification naming and packaging, as well as for SIP formats that allow the easy extraction of included software or emulator specifications (if the submission of such components is supported).

2.4 Archival Storage

Despite the fact that it discusses the need to transform content over time to retain it in understandable form, this process as outlined in the OAIS mentions nothing about ongoing preservation (except for the subprocess “Migrate Media”).[26] One interpretation of the OAIS model in support of migration-based preservation is (as in the current version of the DSEP) to add a Preservation process that periodically takes an existing AIP from Archival Storage, unpackages its content, performs a migration (conversion) of this content, and resubmits a new SIP to Ingest containing this migrated content.

The emulation-based preservation scheme described here places very little demand on the Archival Storage process, asking only that it perform task-1 and task-4 of Section 2.2 above, which require that the bit streams of all digital documents and their required software and emulator specifications be kept machine-readable and inviolate by copying them verbatim to new media whenever necessary. Emulation-based preservation eliminates the need to perform any processing on stored digital documents over time, requiring only that the bit streams representing their contents and all necessary interpreters of that contents (i.e., software and emulator specifications) be kept intact. Note that this copying process is logically required by any preservation scheme that involves storing digital information, whether converted or not. Furthermore, since the bit streams of software and emulator specifications are physically indistinguishable from those of documents, emulation-based preservation allows Archival Storage to perform its media copying function without regard to the meaning of what is being copied, which should greatly improve the efficiency of this process.[27]

[26] In addition, “Migrate Media” is misleadingly named, since it may very well involve the transformation of content as well.

[27] By comparison, migration-based preservation requires converting the format of every document, in addition to copying converted bit streams to new media as necessary. Although the OAIS implicitly assumes that migration is to be used for preservation, it does not represent this process explicitly. However, as discussed above, the DSEP’s augmented OAIS model does represent this process, by the addition of an explicit Preservation process.

2.5 Data Management

In the DSEP design, this process manages essentially all of the metadata for an AIP, meaning that metadata need not be stored in the AIP itself. This allows metadata to be converted as data management needs, formats, software, and systems evolve or migrate, without having to modify AIPs. Similarly, this also allows linkages among AIPs to be maintained without modifying the AIPs themselves.[28]

In terms of the emulation scheme described in Section 2.2, this process is responsible for performing task-2 and task-3. Task-2 demands that any required explanatory metadata associated with digital documents be kept understandable, possibly converting at least the topmost level of such explanations (sufficient to explain how to read lower levels of explanation) as successive explanatory documentation formats become obsolete.[29]

Task-3 demands that linkages be maintained between preserved documents (in AIPs) and their associated software and required emulator specifications (stored in their own AIPs). As metadata structures, formats, and systems evolve, these linkages may need to be modified so that they continue to denote the appropriate entities. Similarly, metadata describing relationships among software or emulator components, versions, and configurations may need to be maintained.

As is true of the bit stream copying requirement that this scheme places on Archival Storage, both of these tasks have their counterparts in most other preservation schemes.

2.6 Access

Note that the OAIS notion of delivering a document or publication as a DIP is somewhat biased toward a traditional view of publications as static, “rendered” entities. If digital artifacts are recognized as the executable entities that they inherently are, then the DIP must be seen as a way of delivering interactive, executable functionality. This implies that a DIP may need to include—or at least link to—software, an emulator specification, an emulator specification interpreter, and an executable version of an emulation virtual machine, as well as the contents of a document or publication.

In order to access a digital document that has been preserved using an emulation-based approach, the DSEP must either:

[28] However, this requires some care, as discussed in Sections 3.1 and 3.2 below.

[29] As argued in Rothenberg 1995, 1999 (op cit, note 2) such conversions should ideally be “subset-reversible” to allow verifying that they do not result in loss. However, since this explanatory documentation is not part of the document that is being preserved, the possibility of loss resulting from conversion need not be of paramount concern, so long as the converted explanation continues to fulfill its purpose.

1. Access the preserved document by running its original software under emulation, using the current emulation virtual machine on some current computing platform, delivering the executable document to the user;

or

2. Generate a vernacular use-copy of the document, by rendering the document using emulation—as in (1) above—and then extracting a use-copy from the emulation environment running on the current emulation virtual machine. After extracting this use-copy once (for a given epoch of use) subsequent access to the use-copy requires running only current ("vernacular") software. If it is desired to save a use-copy for potential re-use in the near future, Access may cache it; if a use-copy is to be saved for future use as an historical document in its own right, then Access must package it as an SIP to be passed to Ingest.

Since the second of these alternatives relies on the first, the first is the more fundamental of the two and is discussed here in greater detail.[30] The precise mechanism used to implement this first alternative (i.e., accessing a preserved document in its native, emulated form) admits of a number of variations. The simplest approach would be to link each preserved document (via metadata) to a specific version of a specific application program. Both the document and the program would be preserved as AIPs within the DSEP, so this linkage would simply associate the DSEP's identifier for the document AIP with the DSEP's identifier for the Software AIP. In the simplest case, this Software AIP could contain a computer memory load image consisting of the running application program and all necessary support and operating system software that must be loaded for it to run.[31] The software AIP would in turn be linked to an Emulator Specification AIP containing the emulator specification for a specific hardware platform, which can run the load image contained in the Software AIP. Finally, the Emulator Specification AIP would be linked to an emulator specification interpreter, which would be run by means of the following strategy.

Suppose first that the emulator specification in the Emulator Specification AIP is expressed in the current emulator specification language (or any previous specification language for which an emulator specification interpreter exists that runs on the current emulation virtual machine—which should include at least all specification languages that were developed during the tenure of the current virtual machine). Then, since the emulation strategy guarantees that some computer platform in the DSEP is always capable of running the current emulation virtual

[30] For further discussion of the second alternative, see Jeff Rothenberg, *Using Emulation to Preserve Digital Documents,* 2000 (forthcoming).

[31] Note: if a document is bundled with its own software (and possibly emulator), then the Software AIP will be the document's own AIP.

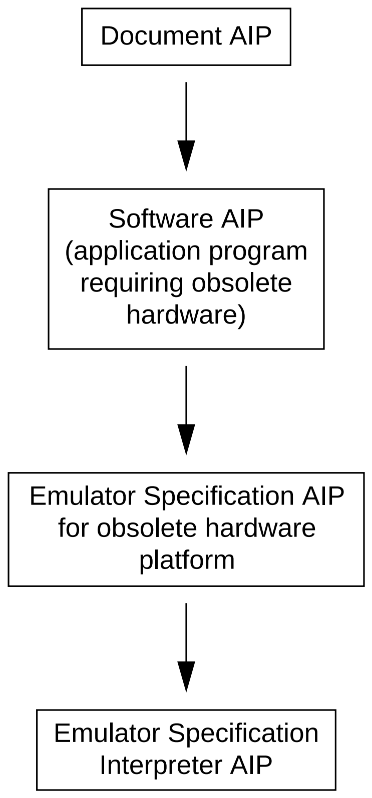

Document AIP
Software AIP (application program requiring obsolete hardware)
Emulator Specification AIP for obsolete hardware platform
Emulator Specification Interpreter AIP

Figure 5: Access via emulation, using linked AIPs

machine, that platform will be capable of running the interpreter for the emulator specification language in question, so the metadata for the Emulator Specification AIP can simply name an appropriate instance of this interpreter. Assuming that the DSEP system always keeps an instance of the current emulation virtual machine running on one of its computers, the DSEP system would then perform the following steps:

Step-1. Load the interpreter designated by the Emulator Specification AIP onto a running instance of the current emulation virtual machine;

Step-2. Extract the emulator specification from the Emulator Specification AIP and run it on this interpreter;

Step-3. Extract the load image containing the application software from the Software AIP and run it on this emulator;[32]

[32] If a document is bundled with its own software, then (as noted above) the Software AIP will be the document's own AIP. In this case, the load image in question will come from the document AIP itself.

Step-4. Extract the document from its AIP and “open” it using the application software, which renders the preserved document.[33]

If the emulator specification in the Emulator Specification AIP is expressed in some older specification language, for which no interpreter exists that runs on the current emulation virtual machine, then an additional step is required. Suppose that the emulator specification language used in this initial Emulator Specification AIP is one for which an interpreter exists that runs on the emulation virtual machine that was in use just prior to the current one (see the left side of Figure 6). Then task-6 in Section 2.2 ensures that there is an emulator specification (S2 in the figure) for this previous emulation virtual machine, written in some emulator specification language (L2) for which an interpreter (I2) exists that runs on the current emulation virtual machine (VM2). In this case, the metadata for the initial Emulator Specification AIP would link to the Emulator Specification AIP (S2) that contains the emulator specification for the previous emulation virtual machine. By hypothesis, this second emulator specification must be written in a specification language for which there is an interpreter that runs on the current emulation virtual machine (as indicated on the right side of Figure 6), so the metadata for this second Emulator Specification AIP would simply name an appropriate instance of that interpreter (I2). The DSEP system would then perform the above steps, using the interpreter designated by the second Emulator Specification AIP (I2) in Step-1 and Step-2: this causes Step-2 to run an emulator of the previous emulation virtual machine (VM1). An additional step would then be performed after Step-2:

Step-2a. Extract the emulator specification from the initial Emulator Specification AIP (S1 in the figure) and run its interpreter (I1) under the emulator of the previous emulation virtual machine (VM1), which was run in Step-2.

For emulator specifications expressed in even older specification languages, whose interpreters run only on emulation virtual machines older than the previous one, this new step is simply repeated.[34] Each iteration of this step runs an emulator of an older emulation virtual machine under the emulation virtual machine that immediately succeeded it, until the original emulator specification is run, at which point the process advances to Step-3.

[33] If a document is bundled with its own software, then (as noted above) the Software AIP will be the document’s own AIP. In this case, opening the document with its application software may be implicitly performed by executing the software in question (i.e., for an executable document); otherwise, this step will be indistinguishable from the more general case, in which the application software, once loaded, must be invoked in such a way as to “open” the document that is to be rendered.

[34] This corresponds to adding additional columns at the left of Figure 6, with each successive previous emulation virtual machine being described by an emulator specification, corresonding to additional instances of the diagonal arrow labelled “*is described by*” in the figure.

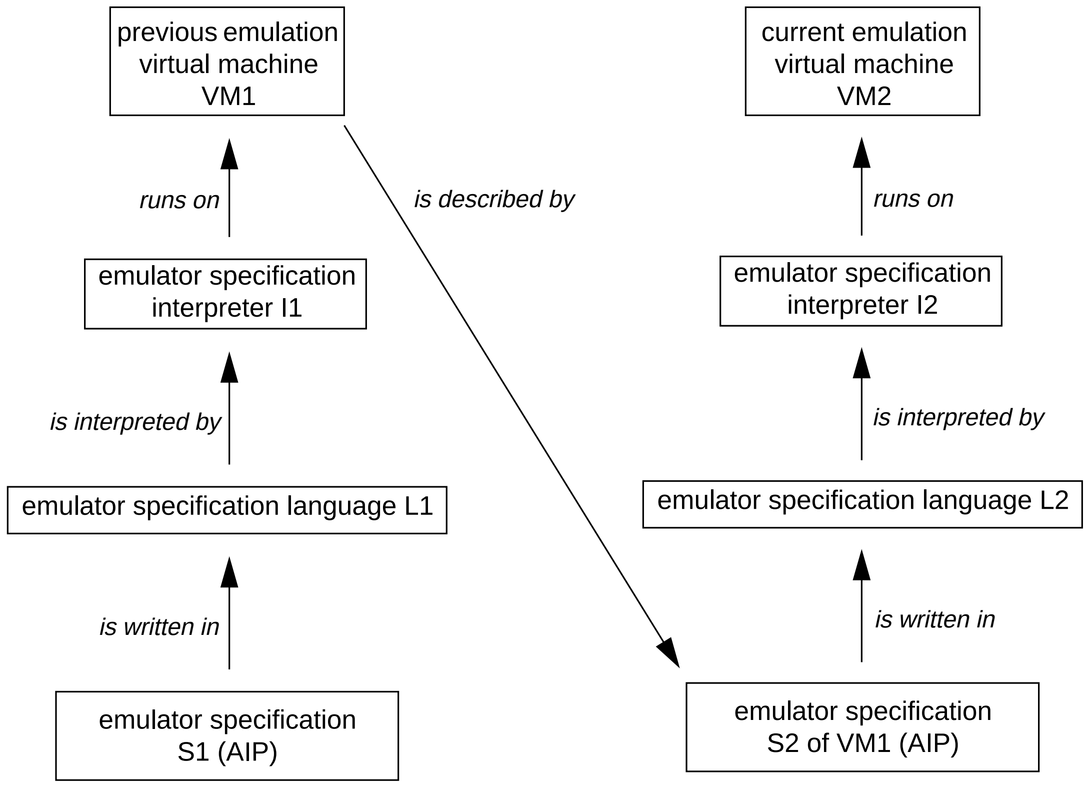

previous emulation virtual machine VM1
current emulation virtual machine VM2
runs on
is described by
runs on
emulator specification interpreter I1
emulator specification interpreter I2
is interpreted by
is interpreted by
emulator specification language L1
emulator specification language L2
is written in
is written in
emulator specification S1 (AIP)
emulator specification S2 of VM1 (AIP)

Figure 6: Emulating previous emulation virtual machines

As noted above, many variations on this basic scheme are possible. For one thing, this approach assumes that a given software application should always run under the same operating system and in the same software and hardware environment, regardless of which preserved document it is rendering. While this may be a reasonable default assumption (since authors of documents cannot generally control these aspects of their readers' environments), there may be cases in which it should be overridden; many applications can be run under various operating systems and on various hardware platforms, and it may be desirable to provide the flexibility for preserved documents to specify such parameters—and for the emulation environment to act accordingly. For example, rather than combining each application and its entire required software environment into a load image to be stored in each Software AIP, software components—or different versions of software for different hardware platforms—could be represented by separate Software AIPs to be loaded as needed. Similarly, emulated hardware platforms could be configured from components represented by separate Emulator Specification AIPs. Such software or hardware configurations would be specified by additional metadata.

2.7 Preservation

Although the emulation-based preservation scheme described in Section 2.2 requires essentially no processing of stored digital documents, it does require generating and maintaining an emulation infrastructure. While some pieces of this infrastructure may be supplied by universities, computer system or software vendors, or other sources, the Preservation process must ensure that the necessary pieces are created and preserved. To this end, Preservation must perform task-5 through task-9 of Section 2.2.

First (task-5 of Section 2.2), whenever a new emulator specification language is developed, a new emulator specification interpreter must be developed for it, to run on whatever is the current emulation virtual machine. Presumably such new languages would be developed by the computer science community or market, in response to the needs of new software or hardware, for which the current emulation specification language had become unsuitable or suboptimal. Any such new language should be designed in concert with an interpreter for that language, if only to demonstrate the utility of the new language (a language without such an interpreter is unlikely to be used). The DSEP need not take a proactive role in instigating the development of such new languages, unless it experiences shortcomings in the current language that affect its ability to preserve digital documents. However, Preservation must ensure that the DSEP has access to an interpreter for the emulator specification language currently in use that runs on the current emulation virtual machine, and whenever a new emulator specification interpreter is developed, Preservation must ensure that someone (either its creator or Preservation itself) packages and submits this new interpreter to Ingest as an SIP, to be made accessible as an AIP in the DSEP for future use.

Next (task-6 of Section 2.2), whenever a new emulation virtual machine is developed, an emulator specification of the current emulation virtual machine must be developed (in the current emulator specification language) so that the new emulation virtual machine will be able to run all previous emulators. New emulation virtual machines might be developed to accommodate—or take better advantage of—new kinds of hardware platforms or to allow running an interpreter for a new kind of emulator specification language. As with new emulator specification languages, a new emulation virtual machine would presumably be developed outside the DSEP (though the DSEP might instigate the development process if it discovers that new hardware will be unable to run the current emulation virtual machine).

At any given moment, the documents currently entering the DSEP will rely on emulator specifications written in the current emulator specification language. This language is capable of describing a hardware platform in such a way that interpreting this description produces an emulator of that platform, which runs on the current emulation virtual machine. Similarly, when a new emulation virtual machine is developed, the current virtual machine (which is itself a virtual hardware platform) must be described in the current emulator specification language, so that interpreting this description on the new virtual machine produces an emulator of the current virtual machine, which runs on the new virtual machine (as illustrated in Figure 6). This allows the new virtual machine to emulate the old virtual machine, so that any emulator specification interpreter that ran on the old virtual machine can now be run on the new virtual machine (under emulation). This in turn allows any previous emulator specification—for any previous hardware platform on which a preserved digital document might depend—to be emulated on the new emulation virtual machine. The Preservation process must ensure that this emulator specification of the previous emulation virtual machine is developed whenever a new emulation virtual machine is introduced. Furthermore, Preservation must ensure that someone (either its creator or Preservation itself) packages and submits this new emulator specification to Ingest as an SIP, to be made accessible as an AIP in the DSEP for future use.[35]

Task-7 of Section 2.2 addresses the need to maintain human-readable documentation for the current emulator specification language and the current emulation virtual machine. Syntactic and semantic documentation of the current emulator specification language must be maintained in human-readable form throughout its tenure so that the language can be used to write new emulator specifications for hardware platforms on which preserved documents will depend. Since (by definition) this language will be in continual use throughout its tenure, its documentation should remain in human-readable form without requiring any explicit effort; however, Preservation must verify that this is the case and perform any actions needed to ensure it. (The format that is used at any given time for such human-readable documentation should be some simple standard form, such as that which is used by Data Management to encode metadata.)

Similar documentation requirements apply to the current emulation virtual machine. At any given time in the future, the DSEP must have one or more computing platforms that are capable of running the emulation virtual machine that is then in current use. If a given emulation virtual machine outlasts the computer platform on which it was originally hosted, it will have to be “ported” to a new platform. While one of the design criteria for an emulation virtual machine should be to minimize the

[35] If a new emulation virtual machine is accompanied by a new emulator specification language (as might be the case, for example, if the development of the new emulation virtual machine were motivated by the desire for a new kind of emulator specification language), then a new emulator specification interpreter for this new language (which runs on the new emulation virtual machine) must also accompany the new virtual machine. In this case, the emulator specification of the current emulation virtual machine should be written in the new emulator specification language, so that it can be interpreted on the new virtual machine without having to write an emulator specification interpreter for the previous emulator specification language to run on the new virtual machine.

amount of effort required to port it to new platforms, this porting process will nevertheless require some understanding of the characteristics and needs of the virtual machine in question, which must therefore be documented in a form that remains human-readable throughout that virtual machine's tenure as the current one. In addition, any new emulator specification language that is designed during the tenure of a given emulation virtual machine will require a new emulator specification interpreter to be written for that virtual machine to interpret emulator specifications written in the new specification language. This will also require human-readable documentation of the emulation virtual machine, to enable such interpreter programs to be written for it.[36]

The Preservation process must verify that each new emulation virtual machine is accompanied by such documentation before it is accepted for use in the DSEP, and it must ensure that any format or content conversions necessary to keep this documentation human-readable are performed. Since (by definition) the virtual machine will be in continual use throughout its tenure, its documentation should remain in human-readable form without requiring any explicit effort; however, Preservation must verify that this is the case and perform any actions needed to ensure it. (The format that is used at any given time for such human-readable documentation should be some simple standard form, such as that which is used by Data Management to encode metadata.)

As noted in Task-8 of Section 2.2, Preservation must also ensure that the current emulation virtual machine always runs on some computing platform that is currently available within the DSEP; whenever a new platform replaces existing platforms in the DSEP, Preservation must ensure that the current emulation virtual machine is ported to that new platform, performing this port internally or contracting it out, if necessary.

Finally (task-9 of Section 2.2), Preservation must create and preserve human-readable explanations for how to use emulation to access preserved digital documents and for how to maintain the emulation infrastructure required for preservation. Unlike the human-readable documentation for particular emulator specification languages or emulation virtual machines, which need not outlast their tenure, this access and infrastructure documentation must be maintained in human-readable form indefinitely since it is the key to using the emulation preservation scheme in the future. This documentation therefore has preservation requirements similar to those of the metadata maintained by the Data Management process: both must be converted as necessary into new human-readable formats over time. This conversion

[36] When an emulation virtual machine is replaced by a newer design, an emulator specification will be developed for the old emulation virtual machine, as discussed above. From this point onward, human-readable documentation for the older virtual machine will no longer be needed, since it need never again be ported to a new platform, and no new emulator specification interpreters will be written for it. Such documentation might be preserved (as a digital document in the DSEP) for historical interest, but this is not required by the preservation scheme itself.

need not be perfect, assuming that the original form of this documentation is not itself considered of historical interest (i.e., in need of preservation); but Preservation must guarantee that this documentation retains its ability to fulfill its explanatory purpose despite being converted.

Note that the model presented here does not assign Preservation the responsibility for acquiring and storing the software required to render preserved digital documents or the emulator specifications of hardware platforms needed to run this software. This responsibility has instead been assigned to Ingest (or Delivery & Capture), since that is the logical place to detect the absence of any missing software or emulator specification components required by AIPs that are about to enter the DSEP. As an alternative to this design, Preservation could be given a more proactive role in identifying and acquiring new software and emulator specifications that are likely to be needed, prior to their being detected as required (and missing) during Ingest.

3. Metadata to support emulation-based preservation

The importance of metadata for library and recordkeeping systems is well accepted. Catalog information, "finding aids" (i.e., information that can help users locate relevant materials), provenance, and administrative information (e.g., location, condition, or usage data) provide additional context that makes a document or record more meaningful, accessible, and useful. In fact, much of the literature on digital repositories (including the OAIS) focuses on metadata—often to the exclusion of the actual documents or records that are to be stored. From the perspective of preservation, the primary metadata issue is to decide what information must be associated with a digital document to facilitate preserving it into the distant future.

As outlined in Section 1 and expanded in Section 4.1, one of the goals of the ongoing experiment described in this report is to incrementally implement an experimental emulation-based approach to preservation within a subset of NEDLIB's evolving DSEP.[37] Other sections of this report (notably Sections 2.2 - 2.7, 5.2, 5.7 and 7) present many of the results of the initial, 1999 iteration of the experiment that should be considered part of the evolving DSEP. This section discusses some of the implications of this iteration of the experiment for metadata requirements and suggests a high-level metadata design for the emulation-based preservation approach that should be implemented within the DSEP.

In light of the analysis presented above, it appears relatively straightforward to store the software required to support emulation within the DSEP (including application and system software load images, emulator descriptions, and emulation virtual machines) as AIPs containing bit streams. Some questions remain concerning how best to represent the required structures for combining these components, but these

[37] Since the DSEP (system) essentially implements the DNEP (depot, or repository), the former term will be used in the remainder of this section, for concreteness.

questions appear (with some exceptions, as discussed below) to affect metadata design more than the design of the AIPs themselves. At the current stage of development of the emulation approach, therefore, the primary aspects of its design that appear to affect the DSEP are those surrounding metadata.

3.1 DSEP metadata considerations for emulation-based preservation

The DSEP will rely on metadata to allow it to access digital records that are preserved using the emulation approach discussed above.[38] The metadata framework presented here consists solely of those items needed to support emulation-based preservation.[39] Additional metadata items, needed for administrative and data management purposes, are discussed in detail in the literature and will not be addressed here.[40] It is important to note, however, that most of the discussion in the literature is concerned with metadata at the conceptual level; while it is important (and often preferable) to keep such conceptual-level discussions free of implementation details, this leads into a dangerous trap in the case of metadata intended to support preservation.

The problem is that conceptual-level metadata design tends to ignore the problem of how the information represented by metadata will itself remain meaningful and usable over time, i.e., how it itself can be preserved. For the most part, this is addressed by assuming that metadata will be "migrated" (i.e., converted as necessary from one standard form to another) as necessary to remain accessible, readable, usable, and meaningful.[41] Since the original form of most metadata need not be considered of historical value, format changes introduced by migration should not be a serious problem for metadata, though corruption is always a danger whenever migration is involved.

---

[38] Note that the term "metadata" as used in the recordkeeping context means data about records and their context. From a data modeling perspective, the proper term for such information is simply "data" rather than "metadata" since the latter term is reserved for information about the data in a database, such as descriptions of database fields in a data dictionary. The term "metadata" is used in the recordkeeping sense throughout this report, but data modelers should be aware that this is a somewhat non-standard usage.

[39] However, the framework presented here should be implemented in an extensible manner so that it can evolve to meet potential future requirements.

[40] See the University of Pittsburgh's metadata specifications derived from the functional requirements of their Reference Model for Business Acceptable Communications (R. J. Cox, "Re-Discovering the Archival Mission: The Recordkeeping Functional Requirements Project at the University of Pittsburgh, A Progress Report," *Archives and Museum Informatics* 8, no. 4 (1994): 279-300; R. J. Cox, "The Record in the Information Age: A Progress Report on Reflection and Research," *Records & Retrieval Report* 12, no. 1 (January 1996): 1-16; and http://www.sis.pitt.edu/~nhprc); the Dublin Core metadata set (see http://purl.org/metadata/dublin_core and H. Thiele, "The Dublin Core and Warwick Framework—A Review of the Literature, March 1995-September 1997," *D-Lib Magazine*, January 1998, http://www.dlib.org/dlib/january98/01thiele.html); the Australian Government Locator Service (AGLS) metadata set (1998-07-27, http://www.ogit.gov.au/aglsindex.html); OAIS (op cit, *note 6*)*; Draft Specification Cedars Preservation Metadata Elements*, 10 December 1999.

[41] This assumption is made explicitly, for example, in the discussion of task-9 in Section 2.7 above.

The remainder of this section and Section 3.2 delve somewhat more deeply into questions of metadata representation in order to identify and analyze some of the problems that have been ignored by most of the metadata literature.

As noted above (in Section 2.1), the DSEP design separates most metadata from the AIPs that are to be preserved, to be stored instead in its administrative computer system.[42] This "metadata store" will be kept on this system in some convenient form, such as in a database or document management system, which is not intended to be a long-lived preservation format. Motivations for this include (a) that administrative and data management metadata will be more readily accessible if they reside in this administrative system than they would be if they had to be extracted from AIPs; and (b) this allows any desired conversion or migration of metadata into different formats to be performed without modifying AIPs themselves.

The problem with including metadata in AIPs, of course, is that the content of AIPs tends to become unintelligible over time, as representations change. This is the crux of the preservation problem, which the emulation scheme described here (and various migration or standards-based schemes described elsewhere) are designed to address. If metadata are considered part of the contents of a publication or document that must be preserved, then they must be handled by some preservation scheme, such as emulation or migration. Yet it seems unnecessarily cumbersome to use emulation just to access metadata about a preserved document, whereas using migration to keep metadata understandable reintroduces the danger of corruption, which emulation was intended to remove. (Ideally, AIPs should be considered immutable, to protect them from inadvertent corruption.)

Fortunately, most DSEP metadata will represent administrative information that is created and maintained by the DSEP itself, which need not be considered part of a submitted publication and so need not be preserved with that publication in its AIP. If a submitted publication contains original metadata that should be preserved for historical purposes, these can be included in the AIP. If such metadata are also of administrative value to the DSEP, they can be copied to the metadata store for convenient use: once copied, the information in the metadata store can be converted or migrated as desired, without needing to modify the original version of the metadata in the AIP and incurring the risk of corrupting it or the remainder of the AIP in the process.

The disadvantage of separating metadata from AIPs is that important information about an AIP may be lost if the metadata store is compromised. AIPs, being designed for long-term storage, will presumably be subject to rigorous access controls and backup procedures, which will be all the more robust if AIPs are kept immutable (since in that case, only new AIPs need be added to previous backups). The metadata store, on the other hand, being part of an administrative system that is in daily use,

[42] That is, the system that the DSEP will use on a daily basis for managing and accessing its assets.

may be far more vulnerable to misuse (whether inadvertent or intentional) and cannot be kept immutable, since it must be constantly updated. Therefore, to the extent that the DSEP relies on information in the metadata store that is not replicated in the AIPs themselves, it may be vulnerable to loss. This is a classic computer system design problem, analogous to the use of file system directories, process tables, and many other situations in which volatile information about more persistent entities is created and maintained for convenient access and modification. One classic solution is to treat this volatile information as a "cache" that can be corrupted or discarded without serious loss because it can always be rebuilt from information stored in the persistent entities themselves.

For the purposes of this analysis, the metadata in the DSEP can be conceptually divided into three categories: (i) information that is of primarily administrative interest; (ii) information that describes individual AIPs; and (iii) information that allows accessing AIPs or links AIPs to each other. If the first of these categories is defined as information which the DSEP can afford to lose without long-term impact (such as usage information about AIPs, use-copies extracted from the native forms of preserved documents, etc.), then it can be allowed to be volatile: any information of this sort that gets corrupted or lost can either be recreated on demand or abandoned.

Information in the second category (such as catalogue information, abstracts, keyword lists, etc., that are created after the AIP's initial ingestion) may be of greater value and less easily reconstructed but is still not crucial to the functioning of the DSEP. Any information of this kind that is of significant value can be added to an AIP, for example, by using a "wrapper" approach, whereby the original AIP is not changed but is simply wrapped in a new AIP that includes this new information.

The third category of metadata is more critical to the functioning of the DSEP. This is analogous to directory information in a file system or index information in a database: if this is lost, it can potentially make it impossible to identify AIPs or to find the crucial linkages among them (for example, between documents and their software or software and its required emulator specifications). The classic solution to this problem (using the "cache" strategy suggested above) would be to include sufficient information in each AIP to allow this metadata to be reconstructed if necessary. However, in order to preserve the robustness of the DSEP, it is crucial to do this without compromising the immutability of AIPs; this requires some thought, since the metadata in question may require periodic modification to remain meaningful.

3.2 Linkages to AIPs: identifiers within the DSEP

The core of this problem involves linkages to AIPs. Such linkages are required by the DSEP both in order to find a given AIP (for example, on the basis of its name or other search criteria) and to allow one AIP to designate another (for example, to link documents to software, software to emulator specifications, emulator specifications to interpreters, and interpreters to emulation virtual machines). The metadata store can provide any number of levels of mapping between human-readable attributes (such as names or keywords) and AIPs, but these mappings must ultimately identify specific AIPs in such a way as to allow accessing them within the DSEP. Furthermore, in the interest of robustness (as argued above), information about the linkages among AIPs must be stored in the AIPs themselves as well as in the metadata store, implying that each AIP may potentially need to identify all other AIPs to which it is logically linked.

There are many alternative ways of identifying entities, but it is useful to factor these into approaches that are intended to be meaningful to humans (i.e., "semantically-meaningful identifiers") versus those that are not (sometimes called "opaque identifiers").[43] The implications of this distinction are far reaching.

Semantically-meaningful identifiers are ones whose face value has some meaning. For example, a descriptive identifier of a building, such as "121 High Street, Santa Monica, 90406" carries considerable information, such as the building's geopolitical location, its geographic relationship to another building designated as "242 High Street, Santa Monica, 90406", and even the fact that it is probably on the North or West side of High Street. Although such identifiers may carry much useful information, they do so by representing some external reality; unfortunately, both that reality and our representations of it can change. (Though street address numbers rarely change, streets are occasionally renamed.) Furthermore, in order to remain intelligible, the form or content of such identifiers may have to change: for example, the spelling or character set used in such an identifier might evolve over time (or, in the example given, the "zip-code" might change from 90406 to 90407-2138, or the U.S. convention of placing street numbers before street names might change, requiring the identifier in question to become "High Street 121, Santa Monica, 90407".)

If semantically-meaningful identifiers are used by the DSEP to designate AIPs, and if such identifiers are stored in the AIPs themselves to provide robustness, then an AIP would have to be changed whenever the form or content of its identifier changed. Moreover, since maintaining links between AIPs would require a given AIP to store the identifiers of other AIPs in addition to its own, this would necessitate updating every occurrence of a changed identifier in every AIP where that identifier was stored; but these occurrences can only be found (efficiently) if "back-pointers" are maintained from each AIP to every AIP that references it. This is precisely the kind of maintenance nightmare that the DSEP's decision to separate metadata from AIPs is presumably intended to avoid.

Fortunately, opaque identifiers do not incur these same problems. By their nature, opaque identifiers are not intended to be interpreted in any way except as designating some specific entity. In the digital case, an opaque identifier is simply a string of bits.

[43] Strictly speaking, describing an identifier as "opaque" merely means that it is used without regard to its possible meaning, typically being handed off to some process without examination; this can be done whether or not it would be meaningful if examiined.

In order to guarantee that such strings can serve as unique identifiers for entities in some collection, it is merely necessary to assign one such bit string to each entity, without repetition, and to store that string in the entity, in some recognizable location of recognizable length (which for concreteness, can be called the "ID field" of the entity).[44] Then so long as the collection of entities allows accessing each entity and extracting its identifier from its ID field, it will be possible to find an entity corresponding to (i.e., containing) a given identifier, namely, by searching the collection for an entity containing that identifier. More efficient access structures (such as indexes) can be constructed from these identifiers, but such structures need not be retained robustly, since they can always be reconstructed from the original collection. A given kind of linkage among entities within the collection (for example, the application program that is required to render a document) can similarly be represented by storing the identifier of the target entity (e.g., the identifier of an AIP containing the application program in question) in a recognizable location of recognizable length (i.e., one of a number of "link" fields) in those entities that link to it.[45] Because such opaque identifiers are never interpreted but are only matched against other such identifiers, they need never be modified within the entities that contain them.[46]

One relaxation of the interdiction against interpreting opaque identifiers that is often considered is to interpret identifiers as numbers in a sequence rather than as arbitrary bit strings. This allows subsets of entities within the collection to be designated "intensionally" by a numerical range of identifiers rather than "extensionally" as a list of individual identifiers. If identifiers are assigned sequentially, then they will correspond (roughly, depending on how many agents are empowered to assign identifiers) to the order in which entities enter the collection. This may be useful for identifying subsets of entities that entered the collection during certain epochs, which might in turn be used to allow entities from different epochs to have different structures or other characteristics. However, even this seemingly innocent change reintroduces some of the problems of semantically-meaningful identifiers. For example, if a later entity is created which for some reason logically belongs in an earlier subset, then the only way it can be inserted into that subset is by changing the identifiers of some of the entities in that subset. (The initial sequence can be defined to leave room between adjacent identifiers to allow such

[44] It is somewhat more flexible to phrase this requirement this way than as a "known" location of "known" length, but in any case, *some* human-readable information about the structure of entities must be retained in order to allow accessing at least some of their fields.

[45] An entity may contain many link fields but only one ID field. Since both link and ID fields contain identifiers, they can be referred to generically as "identifier fields" (whereas an "ID" field can be thought of as an "identification" field, which identifies the entity in which it appears).

[46] The only exception to this occurs if the length of the original identifiers used for a collection turns out to be insufficient to identify all entities in the collection as it expands over time. But even in this case, identifiers need not be modified: they need only be extended. This can be done either by lengthening the fields used to hold identifiers and adding bits onto one or the other end of the original identifiers in those fields (effectively redefining the structure of the entities in the collection) or, more safely, by leaving the original entities intact but "wrapping" them in new entities containing extended identifier fields.

insertions, but any such scheme will break eventually.) In addition, even if sequential identifiers of this sort are used, any subset of entities based on anything other than order of entry into the collection must still be represented by explicit lists of identifiers, so some such list mechanism will have to be provided in any case, casting doubt on the value of the sequential identifier scheme. If epoch-oriented subsets are found to be of sufficient value, alternative mechanisms for defining them might be developed, such as embedding special (opaque) “epoch-identifiers” in entities.

With this discussion in mind, the following sections propose a specific metadata scheme for the DSEP.

3.3 Rationale for DSEP emulation-based preservation metadata

The primary focus of the OAIS concerning the actual preservation of digital artifacts lies in its discussion of Representation Information.[47] In the absence of proven solutions to the preservation problem, the definition of this type of metadata is left relatively unspecified, relying on a “Representation Network” defined as:

> The set of Representation Information which fully describes the meaning of a Data Object. Representation Information in digital forms needs additional Representation Information so its digital forms can be understood over the Long Term.[48]

The recursion inherent in this definition would presumably be terminated by the use of software that can be executed, thereby requiring no further Representation Information beyond the ability to execute it. However, the OAIS cautions against allowing the use of such software to undermine the collection of appropriate Representation Information:

> As a practical matter, software is used to access the Information Object and it will incorporate some understanding of the network of Representation Information objects involved. However this software should not be used as rationale for avoiding identifying and gathering the Representation Information that defines the Information Object because it is harder to preserve working software than to preserve information in digital or hardcopy forms.[49]

The key phrase in this statement is presumably “working software” since it is clearly no harder to preserve software per se than it is to preserve any other digital

[47] Representation Information in the OAIS is part of the Content Information branch of an Information Package rather than being part of its (somewhat misnamed) Preservation Description Information, which is concerned with Provenance, Context, Reference, and Fixity, but *not* preservation per se.

[48] OAIS, op cit, Section 1.7.2, p. 1-12.

[49] OAIS, op cit, Section 2.2.1, p.2-4.

information. Nevertheless, the statement is misleading, since the appropriate comparison is between “working software” and “*working* digital information” where “working” in both cases implies meaningful and usable. Without working software, it is questionable whether working digital information can be preserved at all, as the OAIS itself notes.[50]

The dilemma here is that, on the one hand, human-readable Representation Information (which describes the form of a digital artifact in terms that a human can understand) is both very difficult and expensive to create and very unlikely to be sufficient to interpret any arbitrary digital artifact in the future, as pointed out elsewhere,[51] whereas on the other hand, executable software (which alone is capable of rendering any arbitrary digital artifact in its original, intended form) is not human-readable. The OAIS appears to advocate relying on the former (human-readable Representation Information), whereas the emulation scheme discussed in this report relies on the latter (executable software).

These two kinds of Representation Information (human-readable versus executable) are by no means mutually-exclusive, and in fact, some amount of human-readable documentation is required to support a preservation approach based on software (and emulation), as pointed out in the discussion of task-9 in Section 2.7 above. Keeping additional human-readable Representation Information, however, is irrelevant to an emulation-based preservation scheme, since it is neither necessary nor sufficient for rendering digital artifacts readable over time.[52] Following this reasoning, the metadata design presented below does not include any human-readable Representation Information that is superfluous to the emulation-based preservation scheme; but the omission of such information should not be interpreted as a recommendation against including it. The approach here is simply to present the minimum metadata required to support emulation-based preservation as outlined in this report.

In the OAIS metadata model, Content Representation Information consists of Structure Information and Semantic Information. As pointed out above, emulation-based preservation requires metadata that might be considered Structure Information, in that it links document AIPs to their required application, system, and emulation software. Furthermore, if (as discussed in Section 2.2) software and emulation environments are composed of modular components, each potentially stored in its own AIP, then additional Structure Information may be required to allow configuring appropriate collections of these components in order to render a digital document. However, the linkages between a digital document and its required software can equally be considered Semantic Information, since they specify what

[50] See Section 2.1.

[51] See note 2.

[52] However, adding such information is relatively harmless (except for the added effort that is required to keep it human-readable over time) and may provide support for alternative approaches in the future.

must be done to make the Content Information meaningful. The one type of metadata that seems to fall unambiguously under the category of Semantic Information in the emulation scheme is the overall documentation that explains how to use the scheme to preserve and access documents, as discussed under task-9 in Section 2.7. As noted above, although additional, human-readable Semantic Information may optionally be supplied, it is neither necessary nor sufficient for making a preserved digital artifact meaningful.

3.4 DSEP emulation-based preservation metadata

The DSEP metadata store requires (as a minimum) the following metadata entities and fields (in addition to those required for other archival and administrative purposes) to support emulation-based preservation and access.

In order to ensure the robustness of the DSEP, all mandatory metadata sub-elements describing any entity in the DSEP should also be represented within the AIP for that entity, to allow reconstructing the metadata store from these AIPs if necessary. Since it is desirable for AIPs to be immutable, mandatory elements should ideally not be required to be human-readable, to avoid having to convert their representations in AIPs over time.

Figure 7 shows a notional entity-relationship data model for the required metadata.[53] The model consists of four entities, whose unique identifiers are shown as primary key attributes. Each entity in the metadata model corresponds to an AIP in the repository, identified by the first non-key attribute in each entity. Only the most relevant key and non-key attributes are shown in the figure; the actual non-key attributes of the entities are listed in the following subsections, grouped into what are here (and elsewhere) called elements and sub-elements.

[53] The figure uses IDEF1X graphics with the cardinality of relationships shown numerically (all relationships are of the form 1:1..n, as indicated). Note that all relationships are shown as solid lines, i.e., as "identifying" relationships, which seems the most appropriate representation of the metadata model itself, even though the corresponding AIPs in the repository exist (and are identifiable) independently of each other.

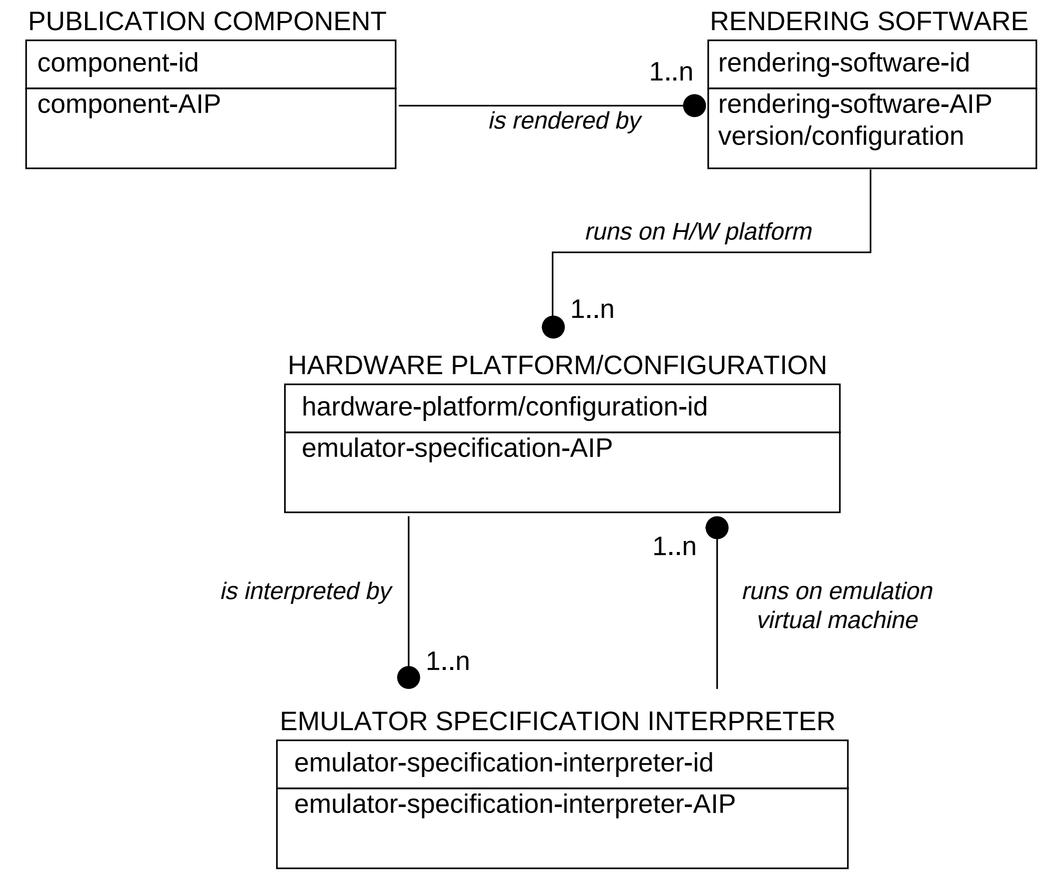

PUBLICATION COMPONENT
component-id
component-AIP
is rendered by
1..n
RENDERING SOFTWARE
rendering-software-id
rendering-software-AIP
version/configuration
runs on H/W platform
1..n
HARDWARE PLATFORM/CONFIGURATION
hardware-platform/configuration-id
emulator-specification-AIP
is interpreted by
1..n
1..n
runs on emulation virtual machine
EMULATOR SPECIFICATION INTERPRETER
emulator-specification-interpreter-id
emulator-specification-interpreter-AIP

Figure 7: Emulation-based preservation metadata model

3.4.1 Entity: PUBLICATION COMPONENT

Data Element: RENDERING SOFTWARE (repeatable)

Requirement: Mandatory

Denotes software capable of validly rendering this component

Sub-elements: [54]

Mandatory: Authenticity of designated software for this publication/component
Mandatory: ID of AIP containing designated software
Mandatory: ID of AIP containing user documentation for designated software
Mandatory: Software invocation string
Mandatory: ID of AIP representing special hardware configuration
(Optional): Name of designated software
(Optional): Component behavior or characteristics using designated software
(Optional): Notes on using designated software on this component

[54] Sub-elements marked "Mandatory" must be included in the design, though their values may be null in certain cases.

Each Publication Component entity in the metadata store corresponds to a publication component preserved as an AIP in the DSEP. This metadata entity enables the emulation scheme to determine which application program must be run to render the component in question.

Each Rendering Software metadata element for a given Publication Component entity designates an AIP containing a specific piece of application software that can be executed to render the given component. Although several versions of a given generic application (e.g., versions of MS Word) may be preserved in the DSEP, each will be preserved within its own Software AIP, so the application designated by an AIP identifier ("ID") in this element will always be a specific version of that application.

As discussed above (see notes 24 and 31) it would be possible to allow an AIP to consist of a Publication Component bundled with its own software (and possibly even emulator specification).[55] In this case (as shown in Figure 8) a Publication Component AIP would simply indicate that it is its own Rendering Software AIP: that is, the value of the field *ID of AIP containing designated software* in the *Rendering Software* metadata element for this Publication Component would be the ID of the component's own AIP.

Since it may be possible to render a given publication component using one of a number of different software applications, this element is repeatable: one such element should be present for each alternative software application that has been verified as capable of rendering the given component with acceptable validity. Some

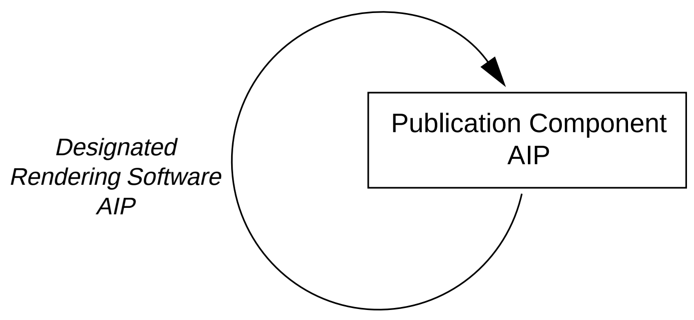


Figure 8: A Publication Component that is its own Rendering Software

[55] Although this is potentially quite inefficient (i.e., if the software or emulator in question are required by any other documents) this possibility is not excluded in the metadata design presented here. In particular, many current CD-ROM publications are of this form.

publications may even have been designed to be rendered by alternative software applications, in which case no one of these alternatives may have a unique claim to being the “native” form for the component. For each instance of this element, the first mandatory sub-element must indicate whether the designated software is one of those applications considered to authentically render the “native” form of the component; alternative software applications may be specified that do not provide this native form but are nevertheless considered useful ways of rendering the component. (In addition to the authenticity specifier itself, an optional sub-element allows describing the behavior or characteristics of the designated software when used on this component, including the ways in which that behavior may differ from the original behavior of the component under its native software. See Section 3.5 for further discussion of representing authenticity in metadata.)

The identifier of an AIP within the DSEP that contains the designated software, as well as the identifier of an AIP within the DSEP that contains user documentation for using this software are also mandatory. Note that the user documentation so designated may be generic or minimal in some cases, but it should be supplied in all cases and should describe the general behavior and characteristics of the software, as well as providing instructions for how to use it. The software invocation string is intended to allow for an uninterpretable string of bits that is to be supplied to the emulation environment to invoke the application in question on the given component.

The final mandatory sub-element (which will often have a null value) allows a component to designate an AIP representing a special hardware configuration on which the designated software should be run to render the given component. Normally, the specified application program will specify its own required hardware configuration, but this sub-element allows a given component to override or amend the hardware configuration that is normally used by a given software application—for example, to specify any specialized peripheral devices, memory requirements, or other hardware needs of the component itself. Since Software AIPs designate hardware configurations themselves, an alternative to this would be to create an AIP representing a specialized version of the designated application, which would in turn designate any such specialized hardware configuration requirements; however, this would proliferate variant versions of Software AIPs representing the same application with different hardware requirements. While the design presented here does not preclude the use of such variant Software AIPs, it provides a more efficient alternative (at the cost of slight additional complexity in the emulation-based access scheme).

Note that of the mandatory sub-elements, only the authenticity specifier has any inherent meaning: the others are all opaque identifiers from the perspective of the DSEP, whose values need not have any human-readable meaning. The authenticity specifier allows the use of alternative software applications to render publication components, even if these applications do not correspond to the native form of the component: if this is not allowed (i.e., if all alternative applications are required to be “native”), then no authenticity specifier is needed.[56] Note, however, that if this sub-element is included in the design, it must retain some meaning for users, implying that it may have to be converted periodically: it therefore represents a slight but singular violation of the principle of immutability of AIPs, suggesting that its value should be carefully weighed. One alternative is to eliminate the concept of using non-native applications to render components, but this seems overly restrictive. Another approach would be to allow such non-native applications to be used for a component but not to represent those applications within the component AIP itself. Under this approach, component AIPs would designate only native applications, and the authenticity specifier would become an Optional sub-element of the component metadata rather than a Mandatory one. A non-native alternative application could then be specified for a component by designating it in a Rendering Software metadata element for that component having an Optional authenticity specifier. The software designated by any metadata element having such an authenticity specifier would be considered an ephemeral part of the metadata store, since the ID of the Software AIP designated by that element would not be represented in the component's AIP. However, any such relationships between components and non-native software could be made non-ephemeral by preserving the metadata elements representing them in AIPs of their own (i.e., AIPs representing aspects of the metadata store that are considered worth preserving).[57]

Optional sub-elements are not strictly required in order for emulation-based preservation and access to work. This implies that they may be allowed to be corrupted during migration/conversion, and need not be reconstructed if lost; however, they provide sufficiently helpful information that they should be preserved in human-readable form if it is cost-effective to do so.

Note in particular that the name of the designated software is considered optional. The emulation scheme does not rely on knowing the name of the application that a component requires, since it does not attempt to understand the format of a component or find an appropriate application for rendering it based on a description and understanding of that format. Rather, it simply links a component to a piece of

[56] There is a difference between an application's being capable of "authentically" rendering a component versus its being the original, "native" application that was intended to be used for the component (or one of several alternative applications, each having equal claim to being the "native" application for the component). For example, future viewers or other programs may be capable of rendering a given digital format correctly (and perhaps even faithfully reproducing all interactions of which a native application was capable) without being one of the original, native applications that were (or could have been) used on the component. Whereas the set of native applications for a component of a given digital format is a closed set that is defined at the creation (or during the original, active lifetime) of the component in question, the set of all future applications that may be capable of authentically rendering a component is open-ended. The authenticity specifier must minimally indicate whether the designated application is considered a native application for the component in question, but it may optionally be used to indicate that other, non-native applications are considered capable of rendering the component authentically, despite the fact that they are not native applications for the component.

[57] The metadata contents of such AIPs could themselves be rendered at any time in the future by using the emulation-based preservation scheme to associate appropriate preserved application software with each such AIP, just as for any other AIP.

executable software that will render it appropriately. Similarities among the digital formats of different components are represented implicitly by the fact that their Rendering Software metadata elements will all designate similar (or identical) sets of application software AIPs. An alternative would be to use the names of format types to select appropriate software for each format; but the design proposed here is more direct and less dependent on human-readable metadata.

3.4.2 Entity: RENDERING SOFTWARE

Data Element: HARDWARE PLATFORM/CONFIGURATION (repeatable)
Requirement: Mandatory
Denotes a hardware platform/configuration capable of executing this software
Sub-elements:
Mandatory: ID of AIP containing emulator specification of designated hardware
Mandatory: ID of AIP containing user documentation for designated hardware
Mandatory: Emulator invocation string
(Optional): Description of designated hardware platform/configuration
(Optional): Notes on using designated hardware to run this software

Data Element: VERSION/CONFIGURATION
Requirement: Optional
Describes the specific version/configuration of this program
Form or Value set: Text

Each Rendering Software entity in the metadata store corresponds to a Software AIP in the DSEP containing a specific application program, embedded in the software context necessary to run it (i.e., corresponding to a “load image” containing all required system software). This metadata entity enables the emulation scheme to determine which hardware emulator must be run in order to execute this application program.

Each Rendering Software entity represents a specific version/configuration of the given program, so the Version/Configuration metadata element is optional, since it merely provides a human-readable description of the relationship of this entity to the generic application program of which it is an instance—and such information is of no interest to the emulation scheme itself, since Publication Components designate such versions directly.

At least one Hardware Platform/Configuration metadata element is mandatory, since it tells the emulation scheme how to execute the program in question. It does this simply by designating an AIP containing an emulator specification for a hardware platform/configuration that is capable of executing the application program as represented in the Software AIP. This element is repeatable to allow specifying alternative hardware platforms or configurations that can run the given program, each represented by a different emulator specification. Alternatively, if there is more than one emulator specification describing a given hardware platforms/configuration, multiple instances of this metadata element can represent these alternatives.

The mandatory emulator invocation string (whose value may be null) is intended to allow for an uninterpretable string of bits that is to be supplied to the emulation environment to supply parameters to the emulator or otherwise condition the emulation process in any specific way that might be required by this program.

3.4.3 Entity: HARDWARE PLATFORM/CONFIGURATION

Data Element: EMULATOR SPECIFICATION INTERPRETER (repeatable)

Requirement: Mandatory

Denotes a program capable of interpreting the emulator specification for this hardware

Sub-elements:

Mandatory: ID of AIP containing the designated emulator specification interpreter
Mandatory: Emulation specification interpreter invocation string
Mandatory: ID of AIP containing user documentation for designated interpreter
(Optional): Notes on using designated interpreter to emulate this hardware

Each Hardware Platform/Configuration entity in the metadata store corresponds to a Software AIP in the DSEP containing the specification for an emulator expressed in some emulator specification language. This metadata entity enables the emulation scheme to determine which emulator specification interpreter must be run in order to emulate the hardware platform/configuration in question.

The Emulator Specification Interpreter element is repeatable to allow for alternative interpreters for a given emulator specification language. The designated emulator specification interpreter in all cases will simply be a Software AIP.

The mandatory emulation specification interpreter invocation string (whose value may be null) is intended to allow for an uninterpretable string of bits that is to be supplied to the emulation environment to provide any necessary parameters to the emulation specification interpreter.

3.4.4 Entity: EMULATOR SPECIFICATION INTERPRETER

Data Element: EMULATION VIRTUAL MACHINE (repeatable)

Requirement: Mandatory

Denotes an emulation virtual machine capable of executing this interpreter

Sub-elements:

Mandatory: ID of AIP containing the designated emulation virtual machine
Mandatory: Emulation virtual machine invocation string
Mandatory: ID of AIP containing user documentation for designated virtual machine
(Optional): Notes on using the designated virtual machine to run this interpreter

Each Emulator Specification Interpreter entity in the metadata store corresponds to a Software AIP in the DSEP containing an emulator specification interpreter program. This metadata entity enables the emulation scheme to determine which emulation virtual machine must be run in order to execute the emulator specification interpreter in question.

The Emulation Virtual Machine element is repeatable to allow alternative virtual machines to be used to run a given emulator specification interpreter. The designated emulation virtual machine in all cases will simply be a Software AIP, which in the general case will be an emulator specification for the designated emulation virtual machine.

The mandatory emulation virtual machine invocation string (whose value may be null) is intended to allow for an uninterpretable string of bits that is to be supplied to the emulation environment to provide any necessary parameters to the emulation virtual machine.

As indicated by the dual relations between the Hardware Platform/Configuration and Emulator Specification Interpreter entities in Figure 7, their relationship is recursive. An emulator specification must be interpreted by an emulator specification interpreter that runs on an emulator virtual machine. But if that emulator virtual machine is obsolete (i.e., is not the current one) then since it is itself a virtual hardware platform/configuration, it will be specified by its own emulator specification (represented by a Hardware Platform/Configuration entity in the metadata model); this emulator specification must in turn be interpreted by another emulator specification interpreter, which runs on a newer emulator virtual machine.

This recursion is terminated when it reaches the current emulator virtual machine, which will, by definition—in addition to being represented by a Software AIP—also exist in running form on one of the DSEP’s computer systems. The emulation scheme will automatically detect this case (by simply recognizing the unique ID of the AIP corresponding to the current emulation virtual machine) and run the emulator specification interpreter under a running instance of the current emulation virtual machine. This corresponds to Step-1 in Section 2.6 in the nominal sequence described in that section (i.e., Step-1 through Step-4).

In all other cases, i.e., where the designated emulation virtual machine is not the current one, the Software AIP representing this older virtual machine will correspond to a Hardware Platform/Configuration entity that describes that older virtual machine. In such cases, the emulation scheme will add Step-2a as described in Section 2.6 and follow the chain of linkages from this new Hardware Platform/ Configuration entity until finding an emulator specification interpreter that can be executed (under however many levels of emulation) on the current emulation virtual machine.

Note that there need be no Emulation Virtual Machine metadata entity per se, since emulation virtual machines are preserved as Software AIPs that are described by Hardware Platform/Configuration entities.

3.5 Example preservation metadata for a CD-ROM publication

This section presents a notional example of how the above metadata might be used to describe one of the CD-ROM publications in the document sample used in the 1999 iteration of the experiment (namely Springer's *Marine planarians of the world*).

Note, however, that the metadata model presented above is intended for a future DSEP repository that relies on emulator specifications, emulator specification interpreters, and emulation virtual machines for preservation. Each of the sample CD-ROM publications used in the initial, 1999 iteration of the experiment, however (as described in the following sections) bundles its software with the publication itself. In a real repository, it would not be feasible to store CD-ROMs in their original forms (since their bit streams would have to be copied to new media as they became obsolete), so such publications would presumably be transferred to other media during Ingest, which might also choose to split apart their publication components and their software, generating distinct AIPs. The CD-ROMs in the sample, therefore, do not represent the forms in which publications are expected to be stored in a real repository. Furthermore, the proposed preservation scheme relies on emulation specifications, interpreters, and emulation virtual machines, whereas the 1999 iteration of the experiment used a specific off-the-shelf emulation program. This makes the *Emulator specification* sub-element of the *Hardware Platform/ Configuration* data element and the entire *Emulator Specification Interpreter* data element inapplicable. Finally, since AIPs do not yet exist in the DSEP, their IDs are replaced in the following metadata fields by appropriate names or other designations. Despite these divergences, it is hoped that the following will give some indication of how the metadata presented above can describe a document preserved by means of emulation.

Entity: PUBLICATION COMPONENT

Data Element: RENDERING SOFTWARE (repeatable)

Sub-elements:

- Mandatory: Authenticity: original
- Mandatory: Designated software: CD (ISBN: 90-75000-14-6)
- Mandatory: User documentation for designated software: CD
- Mandatory: Software invocation string: <double-click-self>
- Mandatory: Special hardware configuration: none

Entity: RENDERING SOFTWARE

Data Element: HARDWARE PLATFORM/CONFIGURATION (repeatable)

Sub-elements:

- Mandatory: Emulator specification: inapplicable
- Mandatory: User documentation for hardware: <VirtualPC 3.0 + Windows95>
- Mandatory: Emulator invocation string: none

Data Element: VERSION/CONFIGURATION
(Optional): "VirtualPC 3.0 + Windows95"

Entity: HARDWARE PLATFORM/CONFIGURATION

Data Element: EMULATOR SPECIFICATION INTERPRETER (repeatable)
Sub-elements:
Mandatory: Emulator specification interpreter: inapplicable
Mandatory: Emulation specification interpreter invocation string: inapplicable
Mandatory: User documentation for designated interpreter: inapplicable

Entity: EMULATOR SPECIFICATION INTERPRETER

Data Element: EMULATION VIRTUAL MACHINE (repeatable)
Sub-elements:
Mandatory: Emulation virtual machine: <VirtualPC 3.0 + Windows95>
Mandatory: Emulation virtual machine invocation string: none
Mandatory: User documentation for virtual machine: <VirtualPC 3.0 + Windows95>

3.6 Additional metadata issues

With the exception of the Authenticity indicator discussed in Section 3.4.1, the scheme proposed above conspicuously lacks any metadata that might support the evaluation of whether and to what extent publications have been authentically preserved.[58] Sections 5.6, 5.7, 5.9 and 7 below discuss authenticity issues in greater depth, along with "validation testing" as a way of operationally evaluating authenticity. As argued there, one reasonable candidate for an "authenticity principle" for a library of deposit is that preserved publications should retain as much as possible of their original functionality and behavior, regardless of what those were or what purpose they are asked to serve in the future. If this position is accepted, it becomes apparent that the best that can be done to authentically preserve digital publications is to preserve the ability to execute them in their original software and (emulated) hardware environments, which by definition retains the greatest possible amount of their original functionality and behavior. Furthermore, although it would be possible to add metadata describing this original functionality and behavior to help verify that it had been preserved correctly in the future, the ultimate futility of such description is what led to the development of the emulation proposal in the first place.[59] On the one hand, it is unnecessary to generate metadata describing the behavior of publications that are preserved by means of emulation, because

[58] Note that "authenticity" in this context is not considered the equivalent of "authentication" which implies the verification that a publication is what it purports to be. Although autentication can also be aided by additional metadata (such as digital signatures or secure hash codes), that is not the focus of this report

[59] That is, if it were feasible to describe the behavior of digital publications well enough to be of use in verifying the authenticity of their preservation, such descriptions would themselves constitute a means of preservation.

emulation will guarantee preserving their behavior as well as can be done, while on the other hand, it is difficult, expensive, and ultimately futile to attempt such description. Generating such descriptions cannot improve the authenticity of the preservation process nor can it materially help evaluate the authenticity of preserved publications.[60]

This argument need not prevent the addition of descriptive metadata intended to support future attempts to verify that digital publications have been preserved authentically; but the omission of such metadata may be a useful way of avoiding the illusion that such metadata can improve the preservation process itself, while avoiding the unnecessary—and significant—cost of trying to describe digital documents.

Finally, note that the metadata model shown above represents a design for a future DSEP repository that relies on emulator specifications, emulator specification interpreters, and emulation virtual machines for preservation. It is not particularly enlightening (and in fact would be rather confusing) to attempt to apply this metadata model to the sample of publications used in the initial, 1999 iteration of the experiment (which is described in the following sections), since this initial iteration bypassed much of this mechanism in order to show the feasibility of the overall approach. For example, as discussed above,[61] each CD-ROM publication in the 1999 sample bundles its software with the publication itself. In a real repository, however, it would not be feasible to store CD-ROMs in their original forms (since their bit streams would have to be copied to new media as they became obsolete). Such publications would therefore presumably be transferred to other media during Ingest, which would presumably also choose to split apart their publication components and their software, generating distinct AIPs. Since the CD-ROMs in the sample therefore do not represent the forms in which publications are expected to be stored in the proposed repository, they are not well suited to the metadata model presented here.

4. Experimental design

This section and the ones following it discuss the methodology and experimental environment that was used to generate the results and recommendations described above.

The overall purpose of the ongoing experiment described in this report is to test the viability of using hardware emulation as a means of preserving digital publications as required by a Library of Deposit, by enabling future computers to run the original software that was required to view (or more generally, to “render”) such publications.

[60] Of course, the software and specifications that the emulation scheme proposes to save to enable publications to be meaningfully accessed in the future could themselves be considered “metadata” describing the publications—but that is not an especially illuminating perspective.

[61] See notes 24, 31, and 55.

In addition, if and as promising techniques are developed in the course of performing this experiment, they are to be implemented incrementally within NEDLIB's evolving DSEP. In experimental terms, this can be viewed as testing the following hypothesis:

Hypothesis

The original hardware environment (processor, display, peripherals, etc.) required to run the original software used to render digital publications can be cost-effectively described with sufficient accuracy to enable the creation of software emulators of that original hardware environment that can be executed by future host hardware environments, and using such emulators to run that original software on future hardware will render saved digital publications in ways that are sufficiently similar to their original renderings to qualify as preserving those publications in the manner required by a Library of Deposit.

This hypothesis must be tested in the context of the following set of experimental variables. Each experimental iteration may choose different values for these variables, thereby testing the above hypothesis in more or less concrete contexts, depending on the resources available and the specific goals of a particular iteration:

Experimental Variables

— Types of digital documents or publications to be preserved

— Criteria for successful preservation of each type of publication in the Library of Deposit context

— Validation tests used to evaluate results against these criteria

— Types of software required to view or render each type of publication

— Original system/configuration requirements of required software

— Timescale of viability of specific emulation approach chosen

— Emulation tools or emulator-building tools and resources employed

— Hardware available for performing/demonstrating emulation

— Degree of reality of non-technical issues to be considered (e.g., intellectual property issues involving software)

Section 4.1 describes the design for the initial iteration of the experiment, which was performed in 1999, whereas later subsections describe the proposed design for future iterations of the experiment.

4.1 First (1999) iteration of experiment

This initial iteration of the experiment was intended primarily to test the experimental environment and procedure themselves; it therefore restricted the values of the experimental variables and set itself a simple goal.

Experimental Variable Conditions

— Document/publication types: small sample to be chosen by KB

— Preservation criteria: to be developed with KB

— Validation tests: alternatives to be identified and analyzed

— Types of software required: to be based on sample document types

— Original system/configuration: as available at KB and RAND

— Timescale: short (i.e., viability on current systems)

— Emulation tools employed: freeware or COTS[62]

— Demonstration hardware: as available at KB and RAND

— Degree of reality of non-technical issues: minimal

Aim

The objective of this first phase of the project was to perform a “base-case” iteration of the experiment by developing an initial experimental environment, materials, and procedures for trying and evaluating emulation-based preservation. In order to evaluate the viability of this experimental framework, identify any missing pieces, and help define the next iteration of the experiment, it was decided to perform a simplified end-to-end run of the entire process. Within the constraints of the project’s limited time and budget, this implied the need to avoid developing any special-purpose software for this first phase, which in turn dictated the use of existing emulation software. Since the bulk of the software-dependent publications of interest to the KB run under the Microsoft Windows operating system (which runs only on Intel-compatible platforms) this determined that we limit ourselves to emulators of the Intel platform capable of running Windows.[63] The number of reliable and

[62] Commercial-off-the-shelf (COTS) or simply “off-the-shelf” (OTS) software refers to programs that already exist and are available without further development.

[63] There exist emulators of many older platforms that run under Windows on Intel platforms, but few of these older platforms are used by digital publications, making them less relevant to a Deposit Library. There are also emulators for older Apple Macintosh platforms that run on Intel platforms (under both Windows and Linux operating systems), but there are few digital publications that can run on a Mac which cannot also run under Windows. All of these facts led us to the conclusion that the most logical demonstration utilizing an off-the-shelf emulator would involve running an Intel-platform emulator on a non-Intel platform.

supported off-the-shelf emulators of this sort is sharply limited at the present time, which constrained our choices, as discussed in Section 5.8. It should be kept in mind while reading this report that the intent of this initial, 1999 iteration of the experiment was not to demonstrate the development of emulation software but rather to show that when such software is developed, it can be used as an effective means of preserving digital publications.

Method

The method used by this first experiment consisted of the following five steps, executed in sequence:

1. Identify initial (1999) and extended ("out-year") requirements for performing the experiment, specifying:

   - Document/publication types
   - Preservation criteria
   - Validation tests and procedures
   - Original system/configuration
   - Demonstration hardware
   - Emulation tools to be employed

   — Develop/acquire initial (1999) instances of each of these.

   — Propose actions required to develop/acquire extended instances of these as well, for out-year iterations.

2. Perform "null" and off-the-shelf (OTS) emulation

   — Use original platform to perform "null" emulation of itself: use this to calibrate the experimental process and validation tests and procedures.

   — Use SoftWindows, WINE,[64] or some similar off-the-shelf emulation environment to run original software to render documents to be preserved on a host platform that differs from the documents' original platform.

3. Apply validation tests and procedures

   — Evaluate results of null case and OTS emulation against validation criteria for preservation.

   — Compare null case and OTS emulated performance and use of documents directly against original performance and use in a "Turing Test"

[64] WINE is a reimplementation of Windows for Unix systems (see http://www.winehq.com/about.html)

  - For null case, vary aspects of original system, such as performance, display characteristics, I/O devices, etc.

  - Possibly "de-tune" performance of original platform in OTS emulation case to "level the playing field" for this test.

— Evaluate the appropriateness and operational viability of the validation tests and procedures themselves, modifying them as necessary.

4. Identify relevant aspects of platforms that must be emulated

— Analyze the results of validation testing to identify those aspects of hardware platforms that it appears necessary to emulate in order to achieve valid preservation results.

— Identify which (if any) of these aspects are specific to particular document types, preservation criteria, uses, etc.

— Identify which of these aspects are not provided by current off-the-shelf emulation products and which (if any) aspects such products provide that are superfluous for preservation purposes.

5. Deliver relevant results for implementation in DSEP

  - Experimental OTS emulation environment
  - Metadata to support preservation via emulation

Results

The results of this first experiment were that we:

— Vetted

  - Initial sample of document/publication types
  - Initial preservation criteria
  - Initial validation tests and procedures
  - Experimental environment
  - Experimental procedure

— Performed initial analysis of aspects of hardware platforms that need to be emulated to satisfy preservation criteria for (at least) the initial sample of document types.

— Identified metadata required to enable specific document types to be preserved using emulation (for implementation in DSEP).

— Determined experimental conditions and goals for the next iteration.

4.2 Proposed second (post-1999) iteration of experiment

The proposed second iteration of the experiment would repeat the initial experiment in a more realistic context. The experimental variables would therefore take on less restricted values, based on the results of the initial iteration:

Experimental Variable Conditions

— Document/publication types: revised, vetted 1999 sample, expanded as necessary to be representative of the full range of problems expected to be encountered in the foreseeable future (i.e., 5-10 years).

— Preservation criteria: revised on the basis of 1999 results

— Validation tests: revised/expanded on the basis of 1999 results

— Types of software required: to be based on revised sample

— Original system/configuration: representative of current KB/NEDLIB practice and projected needs

— Timescale: medium (i.e., beyond current systems, but not requiring extensive transformation or conversion of metadata, emulation specifications, explanatory documentation, etc.)

— Emulation tools employed: to be developed to investigate the feasibility of creating portable emulation environments that can be easily hosted on a range of target platforms to emulate a representative range of original platforms and system configurations

— Demonstration hardware: reasonable range of current platforms

— Degree of reality of non-technical issues: as necessary to prevent future surprises that would invalidate the emulation approach

Aim

Develop emulator specifications for a representative range of original platforms and system configurations, as well as a prototype experimental portable emulation environment that can interpret these specifications (thereby running the specified emulators), and host this environment on a reasonable range of different target platforms to emulate the behavior of the revised, vetted sample of document types required to be preserved, so as to meet the preservation criteria (as embodied in the validation tests) that were vetted by the 1999 iteration.

Method

This second iteration of the experiment would ideally explore both the use of formal languages as a way of specifying emulators and the use of virtual machines or similar approaches for hosting emulators on future computing platforms. It would do this by performing the following steps in sequence:

1. Based on 1999 experience and survey of any recent related work performed elsewhere, develop/acquire post-1999 instances of:

    - Document/publication types
    - Document/publication sample
    - Preservation criteria
    - Validation tests and procedures
    - Original system/configuration
    - Demonstration hardware

2. Explore alternative specification formalisms, including but not limited to:

    - Specification languages (e.g., Z, VDM, B/AMN, Haskell)[65]

    - Hardware Design Languages (e.g., VHDL, Verilog, SIS, VIS)[66]

    - Miscellaneous languages (e.g., Java, Ada, Archelon's MDF)[67]

3. Explore alternative approaches to hosting interpreters for emulation specifications (written in any of the above formalisms) on a range of target platforms, including but not limited to:

    - Java Virtual Machine (JVM)
    - Insignia's Jeode[68]
    - The Tao Group's Elate J-Engine[69]
    - Virtual Virtual Machine (VVM)[70]
    - Archelon's Retargetable Toolset[71]

---

[65] For a comparison of Z, VDM, and B/AMN, see http://www.b-core.com/ZVdmB.html; for further discussion of B, see http://www-scm.tees.ac.uk/bresource/docs/Bhelp2/BPlatform.html. For a good introduction to Haskell, see http://www.haskell.org/tutorial/index.html.

[66] For discussions of VHDL (VHSIC Hardware Description Language) and Verilog, see http://www.optimagic.com/tutorials.html. For a good introduction to VIS (Verification Interacting with Synthesis), see http://www-cad.eecs.berkeley.edu/Respep/Research/vis/doc/cav96/paper/paper.html; for a brief overview of SIS () see http://www-cad.eecs.berkeley.edu/Respep/Research/sis/abstract.html.

[67] Java and Ada are presumably well-enough known to require no specific references. For a discussion of Archelon's Machine Definition File (MDF), see www.archelon.com.

[68] See www.insignia.com.

[69] See http://www.tao-group.com/2/tao/index.html

[70] See http://pauillac.inria.fr/cdrom/projs/sor/projects/vvm/eng.htm.

4. Based on the results of these explorations, choose or develop a formalism and language for specifying portable emulators, along with at least one approach for hosting an interpreter for this formalism on a representative range of target platforms.

   — Acquire/develop tools and capabilities required to be able to:

   - Use the chosen formalism to create emulator specifications for a representative range of original platforms

   and

   - Port an interpreter for this formalism to a reasonable range of currently existing target platforms

5. Develop emulator specifications for a representative range of original platforms and system configurations, using the above emulator specification formalism.

   — Use the post-1999 document/publication sample to derive the attributes of hardware platforms that are required to be emulated.

6. Develop one or more implementations (ports) of a prototype experimental interpreter for the above emulator specification formalism on a reasonable range of existing target platforms.

7. Apply validation tests and procedures

   — Compare documents/publications from the post-1999 sample as they appear in their native environments vs. under emulation

8. Evaluate results

   - Validity of validation tests and procedures themselves
   - Ability of emulation to meet authenticity criteria
   - Feasibility of writing emulator specifications
   - Feasibility of porting an emulation specification interpreter

9. Deliver relevant results for implementation in DSEP

   - Portable emulation specification interpreter
   - Revised metadata to support preservation via emulation
   - Prototype emulation preservation process
   - Additional newly developed techniques, as appropriate

[71] See www.archelon.com.

Anticipated Results

The expected results of the second iteration of the experiment would be to have:

— Vetted

- Revised sample of document/publication types
- Revised preservation criteria
- Revised validation tests and procedures
- Emulation specification formalism
- Portable emulation specification interpreter

— Analyzed the key technical issues involved in using emulation as a practical means of preserving obsolete digital documents/publications.

— Analyzed aspects of hardware platforms that need to be emulated to satisfy preservation criteria for (at least) the post-1999 sample of document types.

— Identified metadata required to enable specific document types to be preserved using emulation.

— Determined experimental conditions and goals for next iteration.

4.3 Proposed third (post-2000) iteration of experiment

The final proposed iteration of the experiment would extend the results of the second iteration to a completely realistic context. It would seek to demonstrate the viability of using emulation to preserve digital documents on a wide range of types of original and host platforms over an indefinite timescale.

Experimental Variable Conditions

— Document/publication types: post-1999 sample (revised if necessary)

— Preservation criteria: as in post-1999 iteration (revised if necessary)

— Validation tests: as in post-1999 iteration (revised if necessary)

— Types of software required: to be based on sample

— Original system/configuration: representative of foreseeable KB/NEDLIB needs

— Timescale: indefinite

— Emulation tools employed: evolutionary revisions of prototype tools developed in post-1999 iteration, extended to investigate ways of dealing with long-term issues such as encapsulation, resistance to corruption of bit streams, creation of appropriate metadata, and reversible conversion of metadata, emulation specifications, explanatory documentation, etc.

— Demonstration hardware: wide range of platforms, to include extreme examples of divergent architectures if appropriate

— Degree of reality of non-technical issues: to be dealt with on a combination of technical, administrative, and policy levels

Aim

Refine and extend the emulator-specification and emulator-environment hosting techniques prototyped in the post-2000 iteration to ensure their long-term viability, and develop a series of "Father Time Scenarios" to test these techniques over simulated extended retention periods.

5. The 1999 emulation experiment

This section describes the first iteration of the experiment specified above, which was performed in 1999 (the "1999 iteration"). As discussed above, the purpose of this first iteration was to develop an initial experimental environment, materials, and procedures for trying and evaluating emulation-based preservation. A simplified "end-to-end" run of the experimental process was performed. This involved developing criteria for selecting a sample of digital documents, selecting such a sample from the KB's own collection, characterizing the documents in that sample to define authenticity criteria and validation tests for them, emulating the native platforms of those documents, running their original applications in this emulated environment, and applying validation tests to evaluate how well emulation reproduced the behavior of the documents in comparison to running their applications on their original platforms. Although these steps were performed in a simplified context, this experience allowed us to evaluate the viability of the experimental process and environment, identify any missing pieces, and help define the next iteration of the experiment.

5.1 A functional model of the experiment

Figure 9 presents a graphical view of the process that was developed to perform the 1999 experimental iteration. Note that this is not to be confused with the emulation functional model (discussed in Section 2) which describes the proposed process by means of which emulation can be used to preserved digital publications. The emulation functional model describes a future preservation process, whereas the functional model presented in this section merely describes the procedure that was used to perform the 1999 iteration of this experiment.

Most of the flows in the figure should be fairly obvious, but there are two feedback flows that may warrant some further explanation. The first of these involves

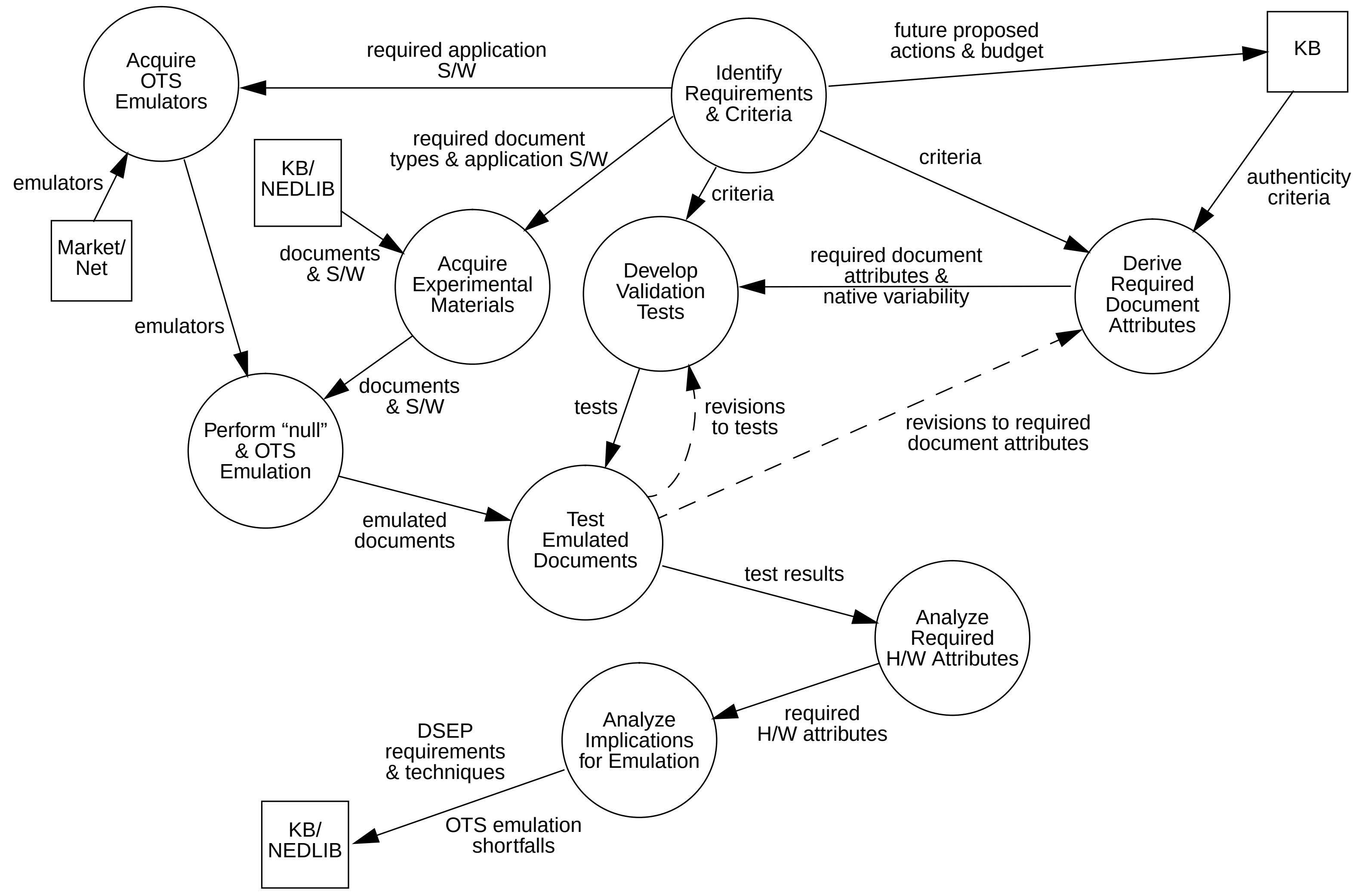

Acquire OTS Emulators
required application S/W
Identify Requirements & Criteria
future proposed actions & budget
KB
emulators
Market/ Net
KB/ NEDLIB
required document types & application S/W
criteria
criteria
authenticity criteria
documents & S/W
Acquire Experimental Materials
Develop Validation Tests
required document attributes & native variability
Derive Required Document Attributes
emulators
documents & S/W
tests
revisions to tests
revisions to required document attributes
Perform "null" & OTS Emulation
emulated documents
Test Emulated Documents
test results
Analyze Required H/W Attributes
required H/W attributes
Analyze Implications for Emulation
DSEP requirements & techniques
OTS emulation shortfalls
KB/ NEDLIB

Figure 9: Functional model of the 1999 emulation experiment

Required Document Attributes, which are initially derived from requirements and criteria, including authenticity criteria, as defined by the KB. These are used to help develop validation tests, but it is expected that validation testing may identify aspects of Required Document Attributes that were initially overlooked. This is represented by the feedback from Test Emulated Documents to Derive Required Document Attributes.

The second feedback loop is also a result of Test Emulated Documents: this feeds back revisions of the validation tests to Develop Validation Tests, based on experience in applying the validation tests.

Finally, note that all of the procedures and tests developed by this process are actually results to be delivered to the KB, although the figure shows only a few of these as explicit outputs.

5.2 Ontology and metadata for the experiment

The following text corresponds to Figure 10: Ontology for 1999 Emulation Experiment. Each object type (class) is described with any component parts indented beneath it, followed by any other (i.e., non-part/whole) associations that emanate from it (that is, which associate it to some other object). Each part/whole component (relation) is followed by an expression [in square brackets] that uses the terms "dependent" or "independent" to indicate whether the existence of the component depends on the existence of its parent. Cardinality expressions <in angle brackets> of the form "1" or "n..m" indicate the number of each such component, where "*" indicates any number. Following any components for an object, an independent indication "{Associations}:" heads a list of all other (i.e., non-part/whole) associations in which the object participates (and which emanate from it). Metadata attributes are indicated in abbreviated form; metadata for selected entities are spelled out in further detail below.

Each Publication consists of one or more Publication Components. Each Publication is characterized by one or more Typical Usage Scenarios, each of which consists of one or more Usage Scenario Tasks that describe how users access and use the Publication.

A Validation Test is associated with one or more Typical Usage Scenarios: that is, each Validation Test compares the results of performing one or more Typical Usage Scenarios using original software on its original Hardware Suite versus an emulated (host) Hardware Suite. A Validation Test Application applies a single Validation Test to one or more Publications, using a single Software Suite and one original Hardware Suite and one emulated (host) Hardware Suite.

Hardware Suites are characterized by a single computer Platform and one or more Peripheral devices. Software Suites are characterized by one or more System

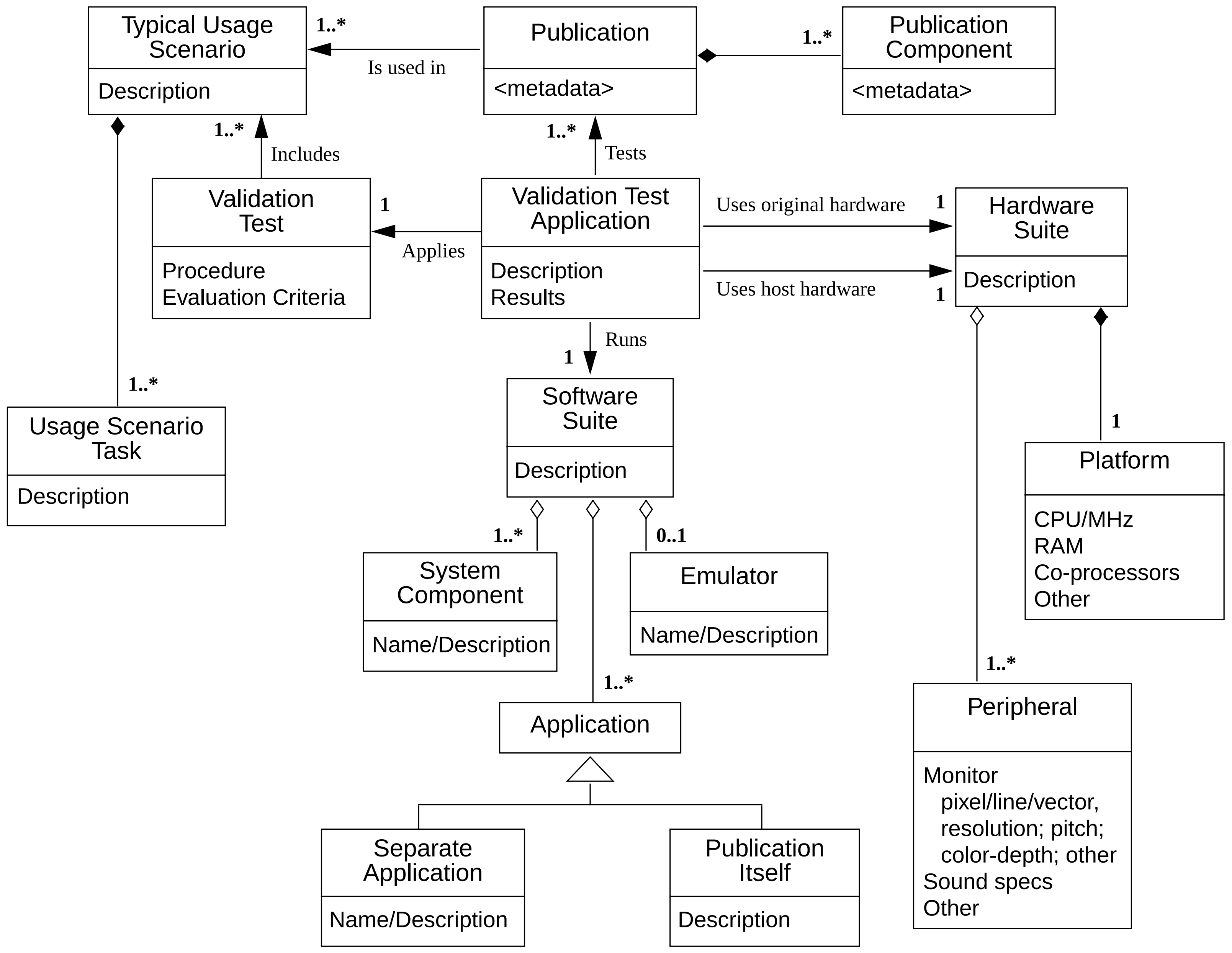


Figure 10: Ontology for 1999 Emulation Experiment

Components, one or more Application programs, and zero or one Emulators (only emulated test cases will use Emulators).

The “specialization” triangle near the bottom of the figure indicates that in some cases, an Application can be considered part of the Publication itself (e.g., for CD-ROM publications that include all necessary software), whereas in other cases it will be a Separate Application.

- Publication (metadata: see below)
  - Publication Component [dependent part <1..*>] (metadata: see below)
  - {Associations}:
    - Is Used In <1..*> Typical Usage Scenario
- Typical Usage Scenario (metadata: description)
  - Usage Scenario Task [dependent part <1..*>] (metadata: description)
- Software Suite
  - System Component [independent part <1..*>] (metadata: name/description)
    - OS, etc.
  - Application [independent part <1..*>]
    - [specialization]
      - separate application (metadata: name)
      - publication itself (metadata: description)
  - Emulation [independent part <0..1>] (metadata: name/description)
- Hardware Suite
  - Platform [dependent part <1>]
    - (metadata: CPU/MHz, RAM, co-processors, etc.)
  - Peripheral [independent part <1..*>]
    - metadata:
      - Monitor (pixel/line/vector, resolution, pitch, color-depth);
      - Sound (specs)
      - etc.
- Validation Test (metadata: procedure; evaluation criteria)
  - {Associations}:
    - Involves <1..*> Typical Usage Scenario

Validation Test Application (metadata: description; results)

{Associations}:

Applies <1> Validation test

Tests <1..n> Publication

Runs <1> Software suite

Uses <1> Original Hardware suite

Uses <1> Host Hardware suite

Metadata for specific entities follows. Note that the metadata specified here is intended to support the experiment itself and need not necessarily correspond to the metadata required by the operational DSEP. In fact, many of the metadata elements shown below were not formally captured for the 1999 iteration of the experiment, though they would still be desirable for future iterations.

Entity: PUBLICATION

Data Element: IDENTIFIER
Form or Value set: URI, ISBN, ISSN, X500, etc.

Data Element: TITLE
Form or Value set: May contain alternative values (as sub-elements)

Data Element: CONTENT DESCRIPTION/SUMMARY
Form or Value set: May include text, images (e.g., thumbnails), etc.

Data Element: AUTHENTICITY CRITERIA
Define requirement for authentic preservation of the publication
Sub-elements (optional):
Publication type: Collection, Report, Letter, Memo, Weblet, etc.
Function description
Relationships to other publications (repeatable):
e.g., part/whole, version, reference, creative (i.e., “based on”), dependency
Behavioral description

Data Element: SOURCE

Data Element: AUTHOR/CREATOR (repeatable)

Data Element: RIGHTS MANAGEMENT/ACCESS/USE CONSTRAINTS AND CONDITIONS

Sub-elements:
- Copyright
- Redaction
- Privacy
- Freedom-of-Information
- (Others as needed)

Data Element: COMMENT (repeatable)

Form or Value set: May include text, images (e.g., thumbnails), etc.

Sub-elements:
- Name/title/role of person making this entry
- Date-and-time of entry

Entity: PUBLICATION-COMPONENT metadata

Data Element: LOGICAL TYPE OF PUBLICATION/COMPONENT

Form or Value set: Physical, IMT (Internet Media Type), ISO standard, etc.

Data Element: LOGICAL STRUCTURE OF PUBLICATION/COMPONENT

Enable access to logical digital publications which may consist of multiple components (technology-dependent)

Data Element: LOGICAL DESCRIPTION OF COMPONENT

Sub-elements:
- Description of logical form and nature of component
- Description of logical encoding (e.g., compression, encryption)
- Note (repeatable)

Data Element: TECHNICAL DESCRIPTION OF COMPONENT

Documents the original digital form of the publication prior to ingestion

Sub-elements:

- Type (original, as-ingested, subsequent)
- Explanation of how to access the component using this description
- Applicable standards
- Non-standard digital format, encoding, etc., if applicable
- Documentation/references for interpreting the digital form of the publication
- Reference to intended application software
- Intended application software documentation/references
- Specific application software dependencies and implications
- Reference to alternate acceptable software (e.g., “viewers”) and documentation
- Environment software (e.g., operating system) dependencies and implications
- Intended hardware characteristics
    - Display (resolution, aspect-ratio, color characteristics, etc.)
    - Storage (access/transfer speed and capacity ranges)
    - Intended range/type of input devices
    - (Others as needed)
- Specific hardware dependencies and implications
    - Display characteristics (resolution, aspect-ratio, color characteristics, etc.)
    - Minimum expected processor/storage/display speeds
    - Maximum expected processor/storage/display speeds
    - Specific input device requirements
    - Additional processing dependencies (co-processors, graphics/sound cards, etc.)
    - (Others as needed)
- Alternate acceptable hardware
- Reference to acceptable physical or emulated hardware
- Reference to documentation for physical or emulated hardware
- Note (repeatable)

5.3 Criteria for choosing a digital publication sample

Some thought was given to choosing a sample of digital publications to be used in the 1999 iteration of the experiment, since the validity and generalizability of the results and conclusions of this iteration would depend on having chosen a representative sample of the kinds of digital publications that are important in the deposit library context. The following general criteria were developed for selecting this sample:

1. Choose examples that are representative of the logical formats, interactivity, and types of standardization that are of interest to a library of deposit;

2. Choose publication types that are representative of a library of deposit’s holdings in terms of relative importance and frequency (this should ideally be based on functional categorization of publications, but if this proves infeasible, it can be based on “publication categories” recognized by the library);

3. Emphasize what are expected to be the most challenging cases, but include at least some of all other important cases;

4. Choose as small a sample as possible that covers enough categories and includes enough examples of each category (chosen to be representative of that category) to prevent excluding any important cases or inadvertently being misled by non-representative cases.

This sample should be winnowed from a larger set of possible candidates by performing the following steps:

a) Choose and acquire 3-5 examples of what appear to be fairly representative members of each category;

b) Compare and analyze these examples to find the minimal number of each category that appear to exhibit all salient attributes, behaviors, and characteristics of the entire group of examples for that category.

c) As a result of this analysis, make a final evaluation of what appear to be the most challenging categories.

This selection process should produce a minimal number of examples that nevertheless provide some confidence that the sample has not excluded important characteristics of any category. The final number of examples that results from this process might be anywhere from 1-5 for each category. The process should also produce a prioritization for the sample, based on tackling the most challenging cases first.

5.4 Sample digital publications used

Using the above criteria, a sample of CD-ROM publications and "online" (journal) publications were chosen by the KB for the sample of digital publications to be used in the 1999 iteration.

The CD-ROM publications chosen for the sample were:

1. Title: Marine planarians of the world

   Year: 1996

   Author: R. Sluys

   Publisher: [Heidelberg etc.: Springer]

   Edition: Version 1.0

   Set up by Expert-center for Taxonomic Identification, Amsterdam

Series: The world biodiversity database

1 CD-ROM (Windows)

System requirements: IBM compatible PC, 80486 processor, 8 Mb RAM, Windows 3.1 or Windows 95, VGA color monitor, CD-ROM player

ISBN: 90-75000-14-6

2. Title: Cultivation and farming of marine plants

   Year: 1997

   Author: A.T. Critchley, M. Ohno (eds)

   Publisher: [Heidelberg etc.: Springer]

   Edition: version 1.0

   Set up by Expert-center for Taxonomic Identification, Amsterdam

   Series: World biodiversity database CD-ROM series

   1 CD-ROM (Macintosh & Windows)

   System requirements: IBM compatible PC, 80486 processor, 8 Mb RAM, Windows 3.1 of windows 95, VGA 16 bit color monitor, quad-speed CD-ROM player, sound card compatible with Windows

   System requirements: Macintosh computer with 68040 processor or PowerPC, MacOS version 7.0; 8 MB RAM; 14” 16-bit color monitor, quad-speed CD-ROM player.

   ISBN: 3-540-14549-4

3. Title: Paranasal sinuses & anterior skull base

   Year: 1993

   Author: Berend Hillen

   Publisher: Amsterdam [etc.]: Elsevier Science Publishers

   Series: The head & neck; I (Elsevier’s Interactive anatomy ISSN 0929-2225; vol. 1-I)

   1 CD-ROM, 1 CD-I

   System requirements: IBM compatible PC, 33 MHz 486 CPU or better; 4 MB internal memory; MS Windows 3.1; 5 MB available on hard disk for installation; CD-ROM player with adequate access time (less than 400 milliseconds); accelerated 256 color VGA adapter compatible with MS-Windows (resolution 640x480); mouse or other pointing device compatible with MS Windows.

System requirements: CD-I: Philips CDI 220 or the portable CDI 360 player with the Digital Video (DV) cartridge; TV with scart connector, 6 inch color LCD-screen.

ISBN: 0-444-89985-5

4. Title: Encyclopedia of neuroscience

   Year: 1999

   Author: George Adelman, Barry H. Smith

   Publisher: Amsterdam [etc.]: Elsevier

   Edition: 2nd rev. and enlarged ed

   Based on print edition: Elsevier's encyclopedia of neuroscience; first print: Boston: Birkhaeuser, 1987

   1 CD-ROM (Macintosh & Windows)

   System requirements: IBM compatible PC, 486 DX66 or better processor, 8 Mb RAM (16 MB recommended), Windows 95, color monitor recommended, double speed CD ROM drive (quad-speed recommended), VGA video card (SVGA recommended); 8 bit sound card; MS mouse or compatible; MS supported printer.

   System requirements: Macintosh; 68030 or better processor (Power Macintosh recommended), MacOS supported printer; 8 MB RAM (16 MB recommended); system 7 or better (7.5 recommended); double speed CD-ROM drive (quad-speed recommended)

   ISBN: 0-444-81612-7

5. Vraag & Aanbod Handelswijzer

   Year: 1998

   Author: Stichting Kwaliteitsdienst-KDI, Rotterdam

   Publisher: Deventer: Kluwer

   Edition: 1999

   1 CD-ROM

   ISSN: 1566-9750

6. Title: Autotechnisch handboek

   Year: 1999

   Corporate author: Kluwer BedrijfsInformatie. Redactie/Development team

   Publisher: Deventer: Kluwer

Edition: version 1.1999

2 CD-ROMs

Publication in two parts: 1: A t/m N. 2: O t/m Z

System requirements: Windows 3.1 or 95

ISBN: 90-201-3011-0

In addition, the following “online” journal articles (available in Adobe PDF format) were included in the sample:

7. Journal title: Agricultural water management

   Volume 38, issue 1, 1 October 1998, page 45-57

   Authors: Mishra, A., Ghorai, A.K., Singh, S.R.

   Title: Rainwater, soil and nutrient conservation in rainfed rice lands in Eastern India

   Keywords: Rainfed rice, Rainfall excess, Water balance components

   Issn: 03783774

   Publisher: Elsevier

   Place of publication: Amsterdam

8. Journal title: Tourism management

   Volume 17, issue 6, September 1996, page 425-432

   Author: Sindiga, I.

   International tourism in Kenya and the marginalization of the Waswahili

   Keywords: tourism, Waswahili, Islam, Kenya

   Issn: 02615177

   Publisher: Elsevier

   Place of publication: Guildford [etc.]

9. Journal title: Minds and machines

   Volume 7, issue 3, 1997, page 365-385

   Authors: Stuart a.Eisenstadt, Herbert A.Simon

   Title: Logic and Thought

   Keywords: formal logic, logic and thought, production systems, inference rules, natural deduction, tautologies

   Issn: 09246495

Publisher: Kluwer Academic

Place of publication: Dordrecht [etc.]

10. Journal Title: Man and world

    Volume 30, issue 4, October 1997, page 431-443

    Authors: Derek K. Heyman

    Title: Dual and non-dual ontology in Satre and Mahyna Buddhism

    Issn: 00251534

    Publisher: Kluwer Academic

    Place of publication: Pittsburgh, Pa.

11. Journal title: Global and planetary change

    Volume 20, issue 4, May 1999, page 281-288

    Authors: Budyko, M.

    Title: Climate catastrophes

    Keywords: Climate catastrophes, Environmental changes, Global carbon cycle

    Issn: 09218181

    Publisher: Elsevier

    Place of publication: Amsterdam [etc.]

5.5 Characterizing the digital publications in the sample

Viewing the CD-ROMs and online publications in the KB's initial sample suggested that we attempt to have someone on the KB staff characterize how each publication in their sample (both CD-ROM and "online" PDF, MrSID, etc.) might be used by a typical user, i.e., developing one or more "typical usage scenarios" for each publication. This was to be used as the basis for developing validation tests to test whether the "preserved" form of a publication can satisfy the intended/expected use (access) of that publication. Note that there might be several typical usage scenarios for a given publication and that a given typical usage scenario might apply to several different publications.

Unfortunately, lack of time and personnel precluded the development of formal typical usage scenarios during the 1999 iteration.

5.5.1 Criteria for characterizing publications in the sample

Since these typical usage scenarios were to be developed in a somewhat ad hoc fashion, we also agreed that the KB would further characterize the functionality and behavior of each publication independently of usage scenario, as a cross-check to guard against our missing any essential or unusual characteristics. This would help ensure that we learned as many general lessons as possible from our sample, since even if a novel behavior were not directly relevant to any of the typical usage scenarios for the publication in which that behavior appears, the behavior (or some similar behavior) might be relevant to the use of other publications not in our sample. For this purpose, we were especially interested in "unobvious" behaviors or aspects of a publication that are not common across many or all digital publications and whose presence is not obviously indicated by a menu item, virtual button, etc. The kinds of "unobvious" behaviors that we identified included (but were not be limited to) the following, which appear to be generally useful for characterizing digital publications that are candidates for preservation:

— Installation: any oddities or difficulties encountered in "mounting" or installing a publication in order to use it.

— References to hardware environment: for example, instructions that appear when using a publication that refer to specific named keys on the keyboard, such as "Enter" or "Escape" (or "ESC") keys (e.g., when trying to interrupt the "demo" mode in one of the ETI/UNESCO publications) or to the pointing device (e.g., as a "mouse").

— Navigation: paging vs. scrolling: use of table-of-contents or other index which user must use to go to sections or pages vs. ability to scroll continuously through the publication; also use of virtual buttons, etc. to navigate, especially any aspect of navigation that is a feature of a viewer used after conversion and which therefore provides a capability that may not have been present in the original digital publication as distributed by its publisher (e.g., for journal articles distributed as TIFF but viewed as PDF).

— Any functionality provided by current viewer that may not have been present in the original digital publication as distributed by its publisher (e.g., for journal articles distributed without text but subsequently OCR scanned).

— Any unusual interactive functionality

- Changing cursor style or behavior

- Ability for user to save preferences, settings, navigation history, bookmarks, annotations, etc., especially across usage sessions.

— Any dependence on original distribution medium, such as references within the content or the interaction prompts of a multi-CD publication to named or numbered disks (e.g., “insert disk 2” appearing as a prompt or “see disk 2” appearing as a cross-reference within displayed text).

— Any interaction between original distribution medium and functionality, such as: need to switch volumes to search across entire database; any discontinuities in sound or video playback due to lack of buffering (OR notable lack of such discontinuities).

— Any dependence on original packaging or information contained in, on, or with that packaging, such as registration or serial number, authorization code, password, etc., especially if required to install or use the publication.

— Any reference within the publication to an external source of information, such as a phone number or URL, especially if required to install or use the publication

— Ability to “print to file” (e.g., save pages, extracted contents, etc. as PostScript, PDF, etc.)

— Active text or “pixel-map” style images

— Animated graphics, sound, video clips, “voiceover” linked to a moving pointer or other visual indication on the screen, where the synchronization of sound and visual behavior are critical to the meaning (e.g., voiceover refers to “this” which is some visual item on screen that is intended to be highlighted as the word “this” is heard).

— Functional or semantic use of color

- Changing the colors of links to indicate whether they have previously been traversed

- Presenting color-coded information that relies on subtle, similar, or named colors (for example, using a textual description or “legend” of color coded information that describes the colors used by name (e.g., “green” or “gold”) without showing how that named color appears on the screen, which could make it impossible to use such information if the colors as displayed do not match their names (e.g., if green appears as blue).

— Behavior of frames (window “panes”), text, figures, etc. as overall display window is resized: does text wrap or extend off edge? If the latter, are horizontal scroll bars provided, and do these disappear if the window shrinks below a certain size? Do figures shrink as the window shrinks, and if so, do they do so continuously or do they make occasional discrete jumps to different sizes?

— Unusual viewing controls, such as the novel “font zoom” feature in the commercial supplier database CD.

5.5.2 Results of characterizing sample

Due to the short timeframe and lack of available personnel, the characterization of the publication sample described above was only partially implemented by the KB. In particular, no formal typical usage scenarios were defined, and the behavioral characterization discussed above was performed informally and only for the CD-ROM publications in the sample, not for the “online” (PDF) publications. Nevertheless, the behavioral characterization of these CD-ROM publications proved interesting and useful, despite its informality, and is presented here essentially verbatim.

1. Marine Planarians of the world

   Volume name of CD is Planarians

   No autorun CD

   Setup.exe in the directory setup must be run to install the application.

   The installation program creates a shortcut in c:\windows\startmenu\programs\ETI, which will start a program directly from the CD.

   The application contains a demo. The only way to stop the demo when running is to hold down the escape key.

   The mouse is the device to browse/navigate the application. There are buttons to click on to go to different sections and there are special navigation buttons.

   To use and run the application the CD is required, QuickTime for Windows must be installed to view videos.

   A default printer in windows must be installed to have print-possibilities.

   The application-window can not be displayed in full-screen.

2. Cultivation and Farming of Marine Plants

   Volume name of CD is Cfmp

   No autorun CD

   Setup.exe in the directory setup must be run to install the application.

   The installation program creates a shortcut in c:\windows\startmenu\programs\ETI, which will start a program directly from the CD.

   The application contains a demo. The only way to stop the demo when running is to hold down the escape key.

   The mouse is the device to browse/navigate the application. There are buttons to click on to go to different sections and there are special navigation buttons.

   To use and run the application the CD is required, QuickTime for Windows must be installed to view videos.

A default printer in windows must be installed to have print-possibilities.

The application-window can not be displayed in full-screen.

3. Paranasal sinuses & Anterior Skull Base, Elsevier's Interactive Anatomy

   Volume name of CD is PSASB

   No autorun CD

   Setup.exe in the root on the CD must be run to install the application.

   The installation program creates a shortcut in c:\windows\startmenu\programs\active library. It also creates a directory c:\psasb

   To use the application under Windows'95 the program update.exe on the 3,52 diskette must be run.

   When starting the application a window with information about Title and Publisher appears which cannot be displayed in full-screen. This window contains two buttons; continue (to continue the startup of the application) and Information (Information menu about the application).

   After clicking on the button Information, a menu with information items appears. The only way to click between the different items is to click on it and click again on the title to close the item. The only way to let this menu disappear is to click on continue.

   The mouse is the only device to browse/navigate the application. There are buttons to click on to go to different sections. (The button-combination Alt-F4 will end the application)

   When typing the F1 button an error appears on the screen, clicking on help with the mouse works properly (?)

   To use and run the application the CD is required, Video for Windows must be installed to view video-screens.

   There is no print-possibility in the application.

   The application-window can not be displayed in full-screen. Only pictures can be displayed full-screen after being zoomed.

4. Elsevier's Encyclopedia of Neuroscience

   Volume name of CD is Encneu_pc

   No autorun CD

   Setup.exe in the root on the CD must be run to install the application.

   The installation program also installs the required programs Netscape Navigator, Shockwave Plug-in en Apple QuickTime for Windows when they are not already installed.

   The installation program creates a shortcut in c:\windows\startmenu\programs\elsevier science.

It also creates a directory c:\ens and Netscape Navigator in c:\programfiles\netscape\navigator.

CoolTalk of Netscape Navigator can be installed also but is not required for the application to run.

Then the Shockwave plug-in and Apple QuickTime is going to be installed.

When starting the application the first time an error appears on the screen which is about an agreement for Netscape. When clicking OK the application continues. Then a standard warning about a bookmark-file for Netscape appears. After clicking OK the introductory html page starts which is the intro-screen of the application.

Because of Netscape Navigator is the main application to run and browse the CD, the full functionality of input-devices in Netscape can be used.

The CD is not required when using the application.

A default printer in Windows must be installed to have print-possibilities.

The application-window can be displayed in full-screen.

5. Vraag & Aanbod Handelswijzer

Volume name of CD: V&a1999-1

No autorun CD

Setup.exe in the root on the CD must be run to install the application.

The installation program creates a shortcut in c:\windows\startmenu\programs\vraag en aanbod handelswijzer. It also creates a directory c:\programfiles\v-en-a

There is a possibility to install more application-data for faster operation.

When starting the application a graphical window with buttons appear. After making a choice by clicking on one of the buttons another window appear, where it is possible to give a search-string or click on an item with the mouse or keyboard.

Text-colors can be changed in the settings of the application. The last changes will be saved.

To use and run the application the CD is required. Video for Windows must be installed to view video-screens.

A default printer in Windows must be installed to have print-possibilities.

The application-window can be displayed in full-screen. The graphical main window and the text-input window cannot be displayed in full-screen.

6. Kluwer – Auto Technische Handboek Op CD-ROM

Volume names of CD are Ath1_991 and Ath2_991

No Autorun CD

Setup.exe in the root on the CD must be run to install the application.

The installation program creates a shortcut in c:\windows\startmenu\programs\kluwer ATH.

The shortcut starts Multimedia Viewer 2.00 from the directory c:\windows\system and opens f:\ath.mvb.

The mouse is the device to browse/navigate the application. There are buttons to click on to go to different sections. The main application is Multimedia Viewer 2.00 so Kluwer ATH has it's functionality.

To use and run the application the CD is required. When Multimedia Viewer is not installed it will be installed automatically by the installation program.

There are two CDs, that contains data for the application. CD1 contains A to N (Niss) and CD2 contains O to V. So it depends on which CD is needed

A default printer in windows must be installed to have print-possibilities.

The application-window can be displayed in full-screen.

5.6 Developing authenticity criteria

As argued elsewhere,[72] preservation without access is meaningless. The primary purpose of preserving publications is to allow future users to discover, locate, retrieve, decipher, view, interpret, understand, and experience them as required. Any digital preservation approach may constrain such access by failing to preserve certain aspects of documents, though saving digital documents in their native forms would entail the fewest such losses. The nature of digital publications makes the problem of preserving them fundamentally different from that of preserving traditional publications. A traditional document is a physical artifact: saving that artifact preserves all aspects of the document that are inherent in its physical being. But there is as yet no accepted definition of digital preservation that ensures saving all aspects of a digital document. Choosing a particular digital preservation method or technology determines which aspects of a document will be preserved and which ones will be sacrificed: any such technological choice has inescapable implications for what will (and will not) be preserved. In the digital case, we must choose what to lose.

Authentic preservation of a digital publication requires that it remain usable in the future in whatever ways are appropriate for it. A digital publication that is "preserved" without being usable in an authentic and valid way has not been meaningfully preserved, i.e., has not been preserved at all.

But what are the "appropriate" ways in which a digital publication must remain usable in the future? In order to ensure that the emulation preservation strategy being tested would avoid the inadvertent loss of any aspects of digital records that

[72] See Rothenberg and Bikson 1999, op cit, note 2 and Jeff Rothenberg, *Preserving Authentic Digital Information*, A Position Paper for the Council on Library & Information Resources, 2000 (forthcoming).

were crucial to ensuring their authentic preservation, we attempted to formulate an explicit “authenticity principle” (that is, a principle for preserving authentic digital publications). Different types of publications may have somewhat different specific authenticity criteria: for example, authenticity criteria for databases or compound, multimedia publications may differ from those for simple textual publications. Yet in all cases the intent of these criteria is to ensure that preserved publications retain whatever aspects of their original behavior, appearance, content, etc. are crucial to their authentic and meaningful use in the future. The purpose of an authenticity principle is to serve as a foundation for determining the specific authenticity criteria that apply to particular types of publications.

One extreme authenticity principle would require that a digital publication retain as much as possible of the exact function, form, appearance, look, and feel that the it presented to its author or publisher, for example, to enable future researchers to evaluate the range of alternatives that were available to the author (and thereby the degree to which the resulting form of the publication may have been determined by constraint versus choice or chance).

A somewhat different principle would require that a digital publication retain the function, form, appearance, look, and feel that it presented to its original intended audience (or readership) to enable future researchers to reconstruct the range of insights or inferences that these original users would have been able to draw from the entity. Whereas retaining all the capabilities that authors or publishers would have had in creating a digital publication requires preserving the ability to modify and reformat that publication using whatever tools were available at the time, retaining the capabilities of readers merely requires preserving the ability to display (or render) the publication as it would have been seen originally.

Alternatively, we might delineate precise and constrained capabilities that future users are to be given in accessing a given digital publication, regardless of the capabilities that its original authors or readers may have had. Such delineated capabilities might range from simple extraction of content to more elaborate viewing, rendering, or analysis, without considering the capabilities of original authors or readers.

As these examples suggest, alternative authenticity principles levy different demands against preservation. For example, the following sequence of decreasingly stringent principles is stated in terms of the relationship between a preserved digital publication and its original instantiation:

— Same for all intents and purposes

— Same functionality and relationships to other publications

— Same “look-and-feel”

— Same content (for any definition of the term)

— Same description

The choice of a given authenticity principle has serious implications for any digital preservation strategy, since it determines which aspects of a publication must be preserved. For example, if it were deemed sufficient to save the textual contents of a digital publication without regard to its visual appearance or its interactive behavior, then preservation would be a fairly simple task. If, however—as appears to be particularly the case for digital publications—meaningful preservation requires the ability to recreate the full functionality and look-and-feel of a publication, then preservation become far more demanding.[73]

Discussions with the KB suggested that an authenticity principle for libraries of deposit should reflect the agreements with publishers under which such libraries operate. Though these agreements (and their interpretations) may vary among the different libraries represented in NEDLIB, it appeared that most publishers who submit a publication for deposit expect it to be preserved in the form in which it was published, without change.[74] This led us to adopt—as a working hypothesis for at least the 1999 iteration of the experiment—an authenticity principle corresponding to the first, most stringent of the alternatives above, namely that a preserved publication should exhibit as much as possible of its original behavior, functionality, and look-and-feel, as well as its content. This authenticity principle was used as a starting point from which we developed specific authenticity criteria for the publications in our sample. For example, the authenticity criteria that apply to preserving a CD-ROM publication involve the preservation of all of the identifiable behavior, functionality, and look-and-feel of that publication (with some reservations, as discussed below).

5.7 Identifying aspects of hardware that must be emulated

If emulation of hardware is to be used to preserve digital publications, it is important to identify those aspects of current hardware environments that are relevant to the proper rendering and use of each type of digital publication to be preserved. This will determine which aspects of current hardware environments must be emulated on future computing platforms in order to allow digital publications of these types to be preserved appropriately.

[73] For further discussion of these issues, see Rothenberg 2000, op cit, note 72.

[74] Although this has been interpreted as prohibiting “reformating” of traditional publications (which is considered tantamount to republication), in the digital case it appears that preservation may require some degree of such reformating, if only in the sense of copying the bit streams representing publications onto new media to prevent their loss. This implies that existing agreements between libraries and publishers may have to adapt to the different reality of digital publication; nevertheless, these agreements seem to form the most logical basis from which to derive authenticity principles for libraries of deposit.

Note, however, that current digital publications are intended to be usable on a variety of platforms and configurations, often using various alternative displays, pointing devices, sound-generation facilities, browsers, viewers, or applications programs, and processors (or versions of processors) of varying types and speeds, sometimes even using alternative operating systems (e.g., MS-Windows or MacOS). This range of variability in the original environments in which digital publications are intended to run defines what might be called a "native range of variability" for such publications. This in turn defines an important criterion for the range of allowable variability in future emulation:

> **The behavior of a digital publication when used under emulation in the future must be considered valid so long as it lies within the *native range of variability* that was originally accepted for the publication.**

Though no formal attempt was made during the 1999 iteration to define the native range of variability of the digital publications in the sample, this criterion was applied informally when analyzing the hardware needs of digital publications. With this in mind, the following procedure was developed (and applied to the sample) for identifying relevant aspects of hardware environments that must be emulated in the future to support preservation:

1. From the Authenticity Criteria for each document type in the sample set, derive a validation test (or tests) for that document type, to test that a document has been preserved properly according to its specified criteria.

2. Identify an initial, hypothetical list of all attributes of each document type in the sample set that must be retained to meet the Authenticity Criteria for that document type. These constitute "Required Document Attributes" for authentic preservation.

3. Validate this hypothetical list of Required Document Attributes for each document type by applying the appropriate validation tests to original instances of documents of that type, running their original software on their original intended hardware platforms. This "null case" application of the validation tests may reveal that additional Required Document Attributes need to be added to the list or that some of the hypothesized Required Document Attributes are superfluous. Conversely, it may reveal weaknesses in the validation tests themselves, requiring their modification. This should therefore be thought of as simultaneously co-validating the tests and the hypothetical list of Required Document Attributes for each document type.

4. Determine the range of distinct original programs (including different versions or variations of each program) that render the Required Document Attributes of each document type within acceptable limits (as determined by further application of the above validation tests if necessary), thereby deriving the

"native range of variability" that is acceptable for each Required Document Attribute of each document type.

5. Similarly, determine the range of original hardware platforms, environments, system configurations, etc. which can run the range of original software for each document type in such a way as to render its Required Document Attributes within acceptable limits (as determined by further application of the above validation tests if necessary), thereby deriving the (possibly extended) native range of variability that is acceptable for each Required Document Attribute of each document type as it might be rendered by all of the potentially different programs identified above when run on all of the potentially different hardware platforms or configurations identified in this step.

6. Collect the resulting set of Required Document Attributes (qualified by the native range of variability for each such attribute) and the corresponding set of original programs for all document types in the sample. Analyze the ways in which these attributes (as rendered by these programs) depend on properties of the hardware on which these programs run. From this analysis, derive those aspects of hardware environments that must be retained or recreated (e.g., via emulation) to retain the Required Document Attributes of all of the document types in the sample within their acceptable native ranges of variability. Call these "Required Hardware Attributes for Authentic Preservation" (or simply "Required Hardware Attributes").

5.8 Off-the-shelf emulation approaches considered and chosen

The 1999 iteration of the experiment was intended to define the experimental process and environment, and the funding available was not sufficient to allow developing emulation software specifically for this purpose. It was therefore decided to use some appropriate off-the-shelf emulation environment to provide a proof-of-concept for the experiment, while avoiding major research and development costs.

Note that we focused exclusively on hardware emulation, rather than on emulating operating systems or application software. Arguments for this choice can be found elsewhere,[75] but these can be summarized by noting that using software to emulate software cannot provide a convincing demonstration of the possibility of using emulation to confer longevity on digital documents, since describing software in sufficient detail to allow its future emulation is far more difficult than describing hardware in equivalent detail. In fact, the latter is a routine aspect of computer science, whereas the former remains an unsolved problem, except in trivial cases.

[75] See references cited in note 2. In addition, this argument is exapnded in a forthcoming publication for the KB by Jeff Rothenberg, entitled *Using Emulation to Preserve Digital Documents*.

A number of currently-available emulation environments were considered for this purpose, including WINE, Insignia's SoftPC, SoftWindows95 and SoftWindows98, and Connectix VirtualPC. However, since WINE essentially emulates operating system behavior rather than hardware, it was eliminated from further consideration. Insignia's SoftWindows95 offered the advantage of being available on both Apple Macintosh and Unix platforms (HP/HP-UX 10.20; Sun/Solaris 2.5 or SunOS 4.1.4; IBM/AIX 4.1.4) as well as having been used for several years (on Macintoshes) by the Principal Investigator of this study, with excellent results. Although the digital longevity laboratory environment at the KB did not contain any Macintosh computers at the inception of this project, it was thought feasible to acquire one in the required timeframe (though this turned out not be the case), and the laboratory did have potential access to one or more Unix platforms being used as servers. For these reasons, SoftWindows95 seemed to offer the most flexibility in terms of targeting the available platforms.

Insignia's SoftWindows98 offered an alternative to SoftWindows95 which would have had the advantage of presumably supporting more recent applications, but SoftWindows98 was not yet available under Unix, which would reduce the targeting options. In addition, the KB's digital longevity laboratory was using Windows95 on a PC platform to run the chosen sample of publications, and compatibility between these publications and Windows98 had not been established. It therefore seemed advantageous to continue to use Windows95 and to choose SoftWindows95 as opposed to SoftWindows98, to ensure compatibility.

Since the Unix platforms available to the KB's digital longevity laboratory were in use as servers, they would not have been ideal experimental platforms: taking these systems down or trying experimental techniques on them would have run the risk of interfering with other users who relied on them as servers. It was therefore decided that the KB would attempt to acquire a Macintosh computer with SoftWindows95 (and a DVD drive) prior to performing validation testing. This would have provided a standalone laboratory system which could be used experimentally without interfering with other users. Ideally, this Mac system would be as fast as possible, to offset the inherent performance difference between software running on its native system versus under emulation. Since Apple had just announced a 500 MHz G4 PowerPC system at the time this decision was made (though this was later downgraded to a 450 MHz machine), this was chosen as the ideal target. Failing this, the KB was confident that it could use an older, lower-performance Mac system (9600); as a last resort, it was decided that the KB would attempt to acquire the Unix version of SoftWindows95 for use on one of their Unix platforms.

An initial assessment of commercial products such as Insignia's SoftWindows indicated that they were capable of emulating original platforms with high fidelity, at least when running on host platforms that are fairly similar to the originals. In particular, if the host platform has similar peripheral devices and interfaces to the original, recreation of the original hardware environment is quite complete.

In order to help identify required aspects of the hardware environment (as discussed in Section 5.7), the KB agreed to undertake the task of characterizing the behavior of sample document/publications (as described in Section 5.5). These characterizations were used to derive specific validation tests, as discussed in the next section; it was hoped that these characterizations, as applied during validation testing, would yield additional insight into the issue of which aspects of hardware environments must be emulated to meet the access needs of the chosen document/publication sample.

The actual approach that was used in the experiment differed somewhat from the intended approach described above. In the first place, Insignia's SoftWindows product was temporarily unavailable for purchase by the KB, since its ownership was being transferred to a different vendor. We therefore used the Connectix VirtualPC emulator (running Windows95) instead, which proved quite satisfactory.

A second problem was that the KB was unable to acquire a new, fast Macintosh computer system in time for our initial experiment, so we were forced to use the older and slower 9600 system which the KB had available. While the speed of this system turned out not to be a problem, its lack of network connectivity did compromise the experiment to a certain degree, as discussed in the next section.

Finally, due to time and personnel constraints, the KB was unable to perform the complete characterization of the sample publications as discussed in Section 5.5; however, this does not appear to have had a serious adverse effect on the experiment.

Despite these minor problems and some additional shortcomings in our experimental procedure, we were able to perform a reasonably convincing initial iteration of the experiment, the results of which were quite encouraging, as discussed in the next section.

5.9 Validation testing in the 1999 iteration

The initial design of the 1999 iteration of the experiment called for performing double-blind "Turing Tests" to see if users could distinguish the results of performing tasks using the digital publications in our sample running on their original platforms versus running on a different platform under emulation. We relaxed this requirement when it became apparent that we would be unable to run emulation on a platform sufficiently faster than the original platform to make up for the expected speed deficit inherent in emulation (typically at least a factor of 3-6). As it turned out, this factor would probably not have invalidated such tests, since the speed of most of the publications in our sample was constrained by factors other than processor speed (e.g., network speed for PDF documents and CD-ROM access speed for CD-ROM publications).

Nevertheless, additional, unforeseen factors would probably have prevented our conducting strict double-blind Turing Tests in any case. In particular, the lack of sufficient time or personnel to develop formal typical usage scenarios (let alone to conduct rigorous double-blind tests) meant that such tests would not have been much more meaningful than the informal, non-blind tests that we conducted ourselves. More formal testing, however, remains a goal for future iterations of the experiment.

We performed our informal tests by setting up two computer systems side-by-side. One of these was the intended “native” system of the publications in question (namely a Pentium-based system running Windows95), while the other was the KB’s model 9600 Macintosh, running Windows95 under Connectix VirtualPC. (Since Windows95 cannot normally be run on a Macintosh computer, this latter configuration was used as a surrogate for some future “host” platform on which an emulator might run.) Figure 11 shows a screen-shot of one of the sample CD-ROM publications running in Windows95, which is in turn running under emulation using VirtualPC on a Macintosh.[76]

One of the first things that became apparent as we conducted our tests on the CD-ROM publications in our sample was that we really needed two copies of each publication to allow running them side-by-side on the two computer systems. Unfortunately, the KB (as a deposit library) is normally given only a single copy of each submitted publication by its publisher, which prevented direct side-by-side testing.[77]

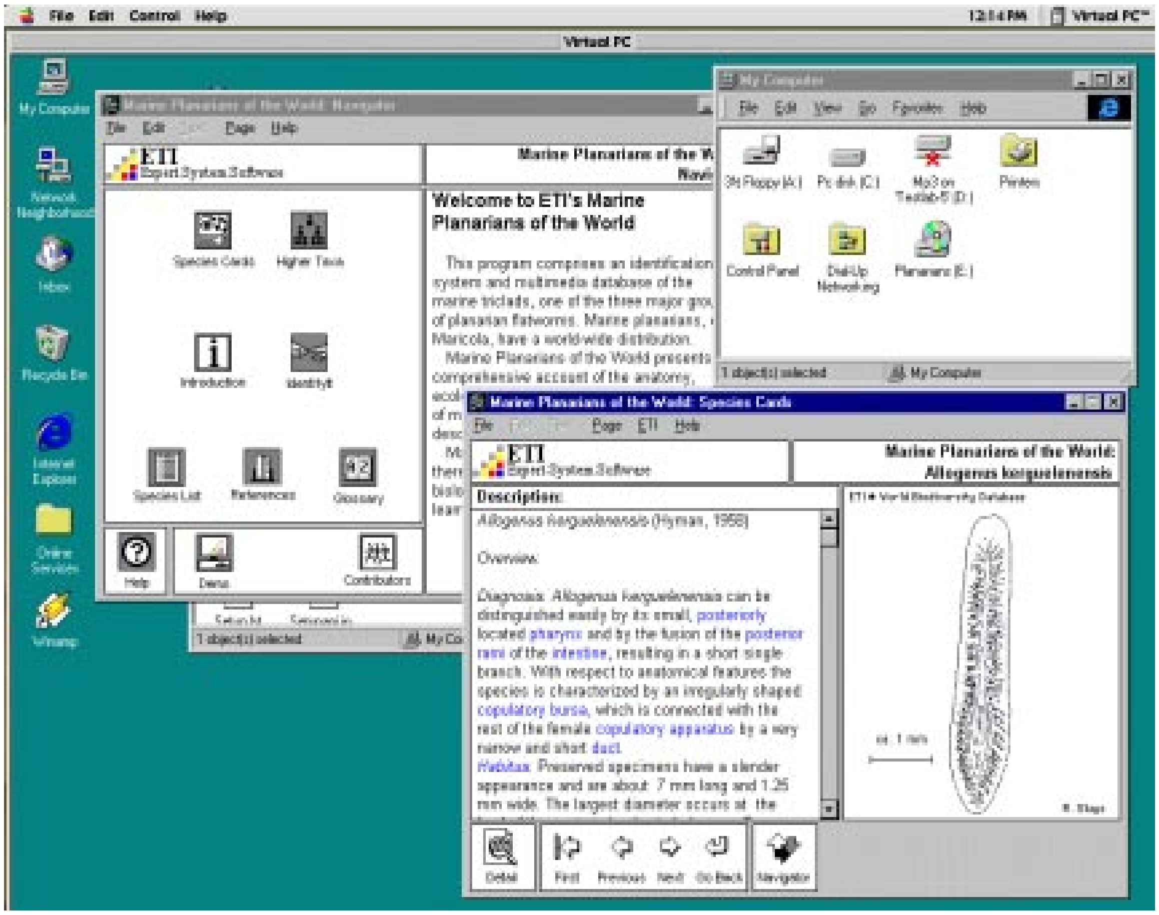
The specific details of our side-by-side comparison runs are of little interest, since the overwhelming (and somewhat surprising) result was that there were no significant differences in the behavior of any of the sample publications. Even the one difference that we were fairly sure we would see, namely a lack of speed under emulation, did not materialize, apparently because the speed with which the applications in question ran on both platforms was constrained by CD-ROM access speed rather than processor speed.[78]

In several cases, when viewing a publication under emulation, we were dismayed to find that the Windows emulator (or the Mac on which it was running) “crashed” or “hung” requiring that it be restarted. However, in every such case, when we tried the

[76] Note the telltale Macintosh menu bar at the top of the screen. The VirtualPC program is running inside a window on the Macintosh desktop, and the Windows desktop appears inside that Macintosh window. The sample CD-ROM publication shown was an extreme example, since running it caused a number of programs to be installed in the (emulated) Windows environment.

[77] In future iterations of the experiment, loadable “images” of the contents of a CD could be used to run the publication on one or both target systems, but this was precluded in the 1999 iteration by the lack of suitable network access for the KB’s Macintosh system. Other possible workarounds, such as copying such loadable images to additional CDs, were precluded by time constraints.

[78] Generalizing from this, since the bit streams of preserved digital publications can be expected to migrate to faster storage media over time, their execution speed under future emulation is unlikely to pose a problem if their performance is constrained by storage-medium access speed rather than by processor speed. In addition, even the expected execution speed degradation due to emulation is likely to be outweighed by the faster processors on which emulators will execute in the future.



Welcome to ETI's Marine Planarians of the World

This program comprises an identification system and multimedia database of the marine triclads, one of the three major gro of planarian flatworms. Marine planarians, Maricola, have a world-wide distribution.
Marine Planarians of the World presents comprehensive account of the anatomy,

Marine Planarians of the World: Allogenus kerguelenensis

Description:
Allogenus kerguelenensis (Hyman, 1958)

Overview:

Diagnosis: Allogenus kerguelenensis can be distinguished easily by its small, posteriorly located pharynx and by the fusion of the posterior rami of the intestine, resulting in a short single branch. With respect to anatomical features the species is characterized by an irregularly shaped copulatory bursa, which is connected with the rest of the female copulatory apparatus by a very narrow and short duct.
Habitat: Preserved specimens have a slender appearance and are about 7 mm long and 1.25 mm wide. The largest diameter occurs at the

Figure 11: A Windows95 CD running under emulation on a Macintosh

same sequence of operations on the publication while running it on its original platform, the native version of Windows (or the Pentium system on which it was running) crashed or hung in exactly the same manner as it had under emulation.

In fact, there were some cases where the emulated environment proved more robust and less troublesome than the native platform, because the latter had been configured in unusual ways or had had non-standard (or updated) versions of system software installed on it—which caused problems for some of the digital publications—whereas the emulated platform had a more standardized and stable configuration, on which all of the sample publications ran without trouble.[79]

In summary, the results of the 1999 iteration of the experiment were surprisingly encouraging as to the potential value of hardware emulation for preserving digital publications over time.

[79] It is worth noting that—unlike most native platforms—an emulated platform can easily be designed to reconfigure itself to a nominal, standard configuration every time it runs a new application.

6. Considerations for future experimental iterations

Future approaches to emulator specification, generation and hosting are being identified and discussed in a forthcoming report that is the output of a companion study performed for the KB,[80] as well as in the experimental design for the "out-year" (post-1999) iterations of this emulation task.

In particular, alternative specification formalisms that are being considered include specification languages (such as Z, VDM, B/AMN, and Haskell), Hardware Design Languages (such as Verilog, VHDL, SIS, and VIS), and various programming or hosting languages (such as Java, Ada, and Archelon's MDF).[81]

Alternative approaches to hosting interpreters for emulation specifications (written in any of these formalisms) on a range of target platforms are also being considered, including the Java Virtual Machine (JVM), Insignia's Jeode, the Tao Group's Elate J-Engine, the Virtual Virtual Machine (VVM), and Archelon's Retargetable Toolset.[82]

7. Validation testing

This section generalizes from the results of the actual validation testing performed in the 1999 iteration (as described in Section 5.9) and offers a recommended procedure for developing and performing validation tests in future iterations.

The development of appropriate validation tests for digital documents and publications is expected to be an ongoing, iterative process. Ideally, "typical usage scenarios" for each sample publication should be developed, and the behavior of each publications should be characterized as described in Section 5.5. The typical usage scenarios for each document in the sample should be used as the basis for developing specific validation tests that will test whether the "preserved" form of a publication can satisfy the intended/expected use (i.e., access) of that publication. There may be several typical usage scenarios for a given publication, and a given typical usage scenario may apply to several different publications.

Validation testing should be applied to each document/publication in the sample (referred to in the following as simply a "document" for convenience). An individual validation test involves each of the following:

- A document

[80] See note 75.

[81] See references for these in footnotes in Section 4.2.

[82] See note 81.

- A set of "typical usage scenarios" that represent the kind of future access and usage expected for the document; each such scenario consists of one or more "scenario tasks" to be performed (where the ordering of these tasks may or may not be relevant)
- The original application software used to view (or "render") that document
- The original system software in which that original application software would normally have run
- An "original" hardware platform/configuration representative of those that would originally have been used to access the document, in which the original application software can be run to view (render) the document and perform all of its typical usage scenarios
- A "host" suite of application and system software (possibly, though not necessarily, the original application software, possibly including emulation software) capable of rendering the document to perform all of its typical usage scenarios [see note below]
- A "host" hardware platform/configuration in which the host software suite can be run to compare the behavior of the document against its behavior when rendered by its original software on its original hardware platform/ configuration in all of its typical usage scenarios
- A comparison between the ability of the document to effectively fulfill each of its typical usage scenarios on its original platform vs. on the host platform

Note: this is worded to encompass validation tests utilizing migration, viewers, and other potential preservation approaches in addition to emulation. For the purposes of testing emulation, the host software suite would be the original software suite plus suitable emulation software to allow that original software to run on the host platform.

A "null" validation test (as opposed to a "real" validation test) is one in which the original and host platforms are the same. In such a test, the behavior of a document on its original platform is compared to itself (i.e., to its behavior on the same platform, running under its original software); this is used to validate the validation test itself (i.e., its procedures and evaluation methods) as well as the typical usage scenarios for the document. Ideally, a null validation test would be conducted in a double-blind experimental setting, where neither user nor observer were aware that the test was null.

In a real validation test, the host platform is different from the original platform, and emulation would be used to run the original software on the host platform. Ideally, a real validation test would be conducted as a double-blind "Turing Test" where neither users nor observers would know whether a given attempt to perform a typical usage scenario on a document was being performed on the original hardware platform or on the host platform. Note that for a fair Turing Test, the performance parameters (e.g., speed) of the original and host platforms should ideally be chosen or adjusted to offset the inherent penalty of emulation; this would “level the playing field” to better simulate the projected use of emulation on future platforms whose presumably higher performance would automatically compensate for this discrepancy. Even if a true Turing Test is not being performed, it may be necessary or helpful to adjust relative speeds if possible, e.g., by running emulation on an inherently faster host platform than the original platform or by artificially slowing down the original platform; if this is not done and the resulting performance discrepancy is too great, it may invalidate the test procedure by interfering with users’ ability to perform or objectively compare the behavior of a document on original and host platforms.[83]

The procedure to be followed to perform any validation test (whether null or real) is:

Validation test procedure test:

i. Choose a single, representative sample document and typical usage scenario and perform “pre-test” steps (1) and (2) on it.

ii. Perform one or more “dry run” null tests, comparing the result of performing the given typical usage scenario on the given document on its original hardware platform against the result of performing it on the same (original) hardware platform to validate the testing procedure itself. Any problem encountered during one of these dry runs should lead to refining the test procedure, which should then be tested by another dry run null test, until the test procedure is deemed workable.

Validation test procedure:

For each typical usage scenario of each document in the sample:

Pre-test:

1. Perform a “dry run” of the scenario using the document on its original platform, to ensure that the “original system” condition of the test environment is working and configured correctly, as well as to validate the scenario.

2. Perform a “dry run” of the scenario using the document on its host platform, to ensure that the “host system” condition of the test environment is working and configured correctly.

[83] However, see a counterargument to this in note 78.

Actual test:

3. Compare the result of performing the given typical usage scenario on the given document on its original hardware platform against the result of performing it on the host hardware platform, ignoring speed differences if not controlling for speed. On both platforms:

   a) Evaluate the ability to perform each task of the usage scenario, as well as the scenario as a whole (e.g., not-at-all, somewhat, mostly, entirely).

   b) Evaluate the difficulty of performing the usage scenario.

   c) Evaluate the appropriateness, authenticity, and usability of the results of performing the scenario.

   d) Record any noteworthy aspects of the test, focusing on differences observed between the behavior of the document on the original and host platforms as well as any interactions between this behavior and any specific characteristics of these platforms.

   e) Evaluate the appropriateness and operational viability of the typical usage scenarios and the validation tests and procedures themselves, modifying each as necessary.

8. Conclusions and Recommendations

The results of this study suggest that using software emulation to reproduce the behavior of obsolete computing platforms on newer platforms offers a way of running a digital document's original software in the far future, thereby recreating the content, behavior, and "look-and-feel" of the original document. The results of the initial iteration of the experiment described above indicate that this approach should work in principle, assuming that suitable emulators for obsolete computing platforms can be hosted on future platforms.

In order to demonstrate that emulation-based preservation can work in practice as well as in principle, it is necessary to develop specification techniques that can describe all relevant attributes of hardware computing platforms in sufficient detail to allow them to be emulated on future platforms. It is also necessary to develop techniques for hosting such emulators on arbitrary future platforms with a minimum of effort. In addition, it is necessary to continue to refine the metadata required to describe digital documents and to link them to the software and emulated hardware environments required to render them in the future. Moreover, it is necessary to ensure that these descriptions can themselves be maintained in human-readable form indefinitely. Finally, it is necessary to continue to identify and refine appropriate authenticity criteria and validation tests for digital documents of various types, to provide a mechanism for evaluating the success of emulation-based preservation (or for that matter, of any other method of preservation).

In order to further explore the potential of emulation as a preservation technique, it is recommended that additional iterations of this experiment be performed, as described above. This would allow more realistic demonstrations of the effectiveness of emulation as a preservation method while developing the key techniques described above. If a series of such experiments is performed as suggested—and if no major surprises are encountered in performing them—it seems likely that a viable, long-term solution to the problem of digital preservation can be developed, using emulation to enable a document's original application software to be run indefinitely.

GLOSSARY

AIP	An Archival Information Package (as defined in the OAIS).
ASCII	American Standard Code for Information Interchange (a common digital character code, used to represent alphabetic, numeric, and punctuation characters in 7-bit binary bytes).
DIP	Dissemination Information Package (as defined in the OAIS).
DNEP	The NEDLIB Depot Nederlandse Elektronische Publicaties (the digital publication repository is to be implemented by the DSEP).
DSEP	The NEDLIB Deposit System for Electronic Publications (which implements the DNEP).
ICT	The Information and Communication Technology department of the KB.
IDEF1X	A U.S. Federal Information Processing Standards Publication (FIPS 184) for data modeling (see http://www.idef.com/ overviews/idef1x.htm).
KB	The Koninklijke Bibliotheek (the National Library of the Netherlands).
NEDLIB	Networked European Deposit Library (a collaborative project of a number of European national libraries, which aims to construct the basic infrastructure upon which a networked European deposit library can be built).
OAIS	The Open Archival Information System.
OCR	Optical Character Recognition (a technique by means of which bit-mapped images of printed characters can be "recognized" to yield character representations, such as ASCII).
OTS	Off-the-shelf (as in Commercial off-the-shelf, or COTS).
PDF	Adobe's Portable Document Format.
SIP	A Submission Information Package (as defined in the OAIS).
TIFF	Tagged Image File Format (a standard for representing digital raster images).
VHSIC	Very High Speed Integrated Circuit
WINE	A freely available reimplementation of Windows for Unix.