

P reservation of the

Jeff Rothenberg

Digital informational artifacts, including records, documents, and data, share a number of core digital capabilities that give them irresistible advantages over traditional paper artifacts. First, they can be copied perfectly, which allows them to be distributed and disseminated widely and accessed remotely. In addition, records and information managers can search their contents, extract content from them, reformat, transform, and process them in ways that are unthinkable for non-digital artifacts. Beyond these core attributes, many digital artifacts possess inherently digital capabilities, such as dynamic, distributed, active, and interactive behavior – facilities that traditional artifacts simply cannot provide.

At the Core

This article examines:

- Emulation in the context of other proposed solutions to digital preservation
- The advantages and challenges of using emulation to preserve digital artifacts
- Two alternative approaches to running emulators on computers in the far future

Yet all these capabilities derive from the fact that digital artifacts are encoded, which – although it makes them understandable to machines – makes them unintelligible to humans without additional interpretation. It is as if they were written in invisible ink. Rendering them human-readable generally requires running programs on computers, especially for those inherently digital artifacts that use complex, executable formats. Furthermore, whereas simple page image artifacts can be printed on paper, inherently digital artifacts cannot be converted to

non-digital form without losing essential aspects of their behavior, not to mention their “look and feel.” This means that digital artifacts must generally be preserved in executable, digital form.

Preserving digital artifacts therefore requires not only saving the bitstreams that represent them but also retaining the ability to interpret those bitstreams properly in the future to recreate their intended behavior. Saving bits is problematic because most

Times

digital storage media become physically unreadable or obsolete rather quickly. But the greater challenge of long-term digital preservation is interpreting bitstreams correctly in the future. Each digital format – corresponding to a given standard or an application program – requires different interpretation, and although a few dozen formats may account for most digital artifacts, there are thousands of other formats whose importance may become apparent only in the future.

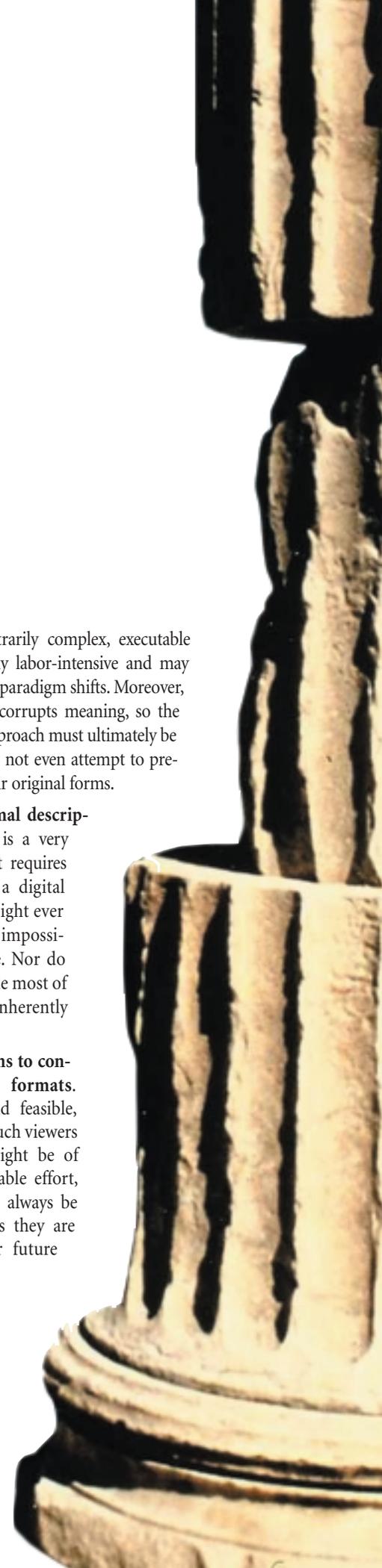
Consider the Options

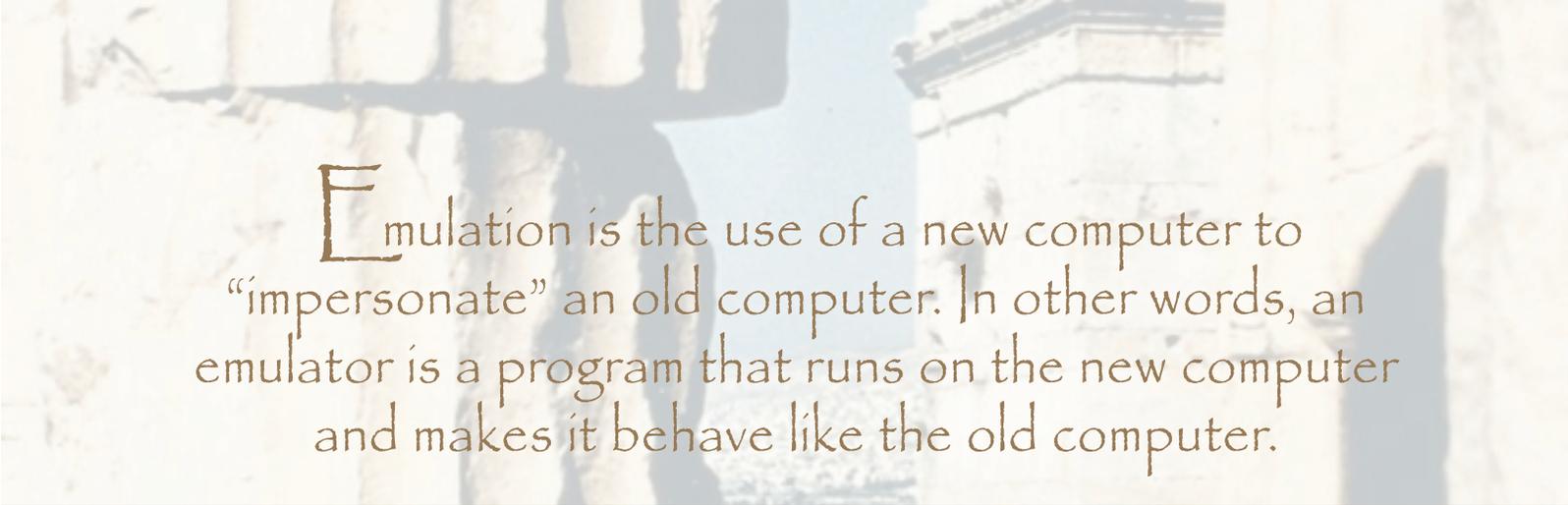
A number of solutions to this problem have been proposed. Among them are:

- **Do nothing.** This argues that most information isn't worth saving anyway.
- **Let the future worry about it (“digital archeology”).** This requires future managers and researchers to bear the cost of understanding whatever they care about. Yet even using sophisticated cryptographic techniques, future users will be far less able to decipher digital artifacts than those who were able to read hieroglyphics during the 13 centuries prior to discovering the Rosetta Stone.
- **Use standard or “canonical” digital formats for everything.** This can help for a decade or so, but in the long term it assumes that future software will be able to render such standard formats. Yet it is reasonable to expect that all such formats eventually become obsolete, and it is impractical to enforce their use in any case.
- **Repeatedly convert artifacts into future formats (“migration”).** This is the obvious choice for records or other artifacts that remain “active” and so must be converted into current forms to be usable. But for the long term – and with the vast majority of artifacts – it is impractical. Converting every artifact

represented in every arbitrarily complex, executable format would be extremely labor-intensive and may not even be possible across paradigm shifts. Moreover, every conversion loses or corrupts meaning, so the cumulative result of this approach must ultimately be gibberish. Finally, this does not even attempt to preserve digital artifacts in their original forms.

- **Replace artifacts by formal descriptions.** In principle, this is a very attractive approach, but it requires formally encoding all of a digital artifact's attributes that might ever be of interest, though it is impossible to predict all of these. Nor do we yet know how to encode most of the behavioral aspects of inherently digital artifacts.
- **Rely on “viewer” programs to continue to render old formats.** Although this may sound feasible, writing and maintaining such viewers for every format that might be of interest requires considerable effort, and their correctness will always be questionable, especially as they are continually rewritten for future computers.
- **Rely on a digital artifact's original software to render it.** Since its original software defines an artifact's format, this is the only way to truly preserve





Emulation is the use of a new computer to “impersonate” an old computer. In other words, an emulator is a program that runs on the new computer and makes it behave like the old computer.

the artifact’s original behavior. (Preserving “reader” software is sufficient for this purpose unless there is a need to preserve a record of the original authoring capabilities that were available.) Although this requires saving application and system software as well as the artifact itself, these are all just bitstreams. Unlike most other approaches, this does not require writing new programs for each format and processing each artifact to be saved. It does, however, require running software long after the original hardware on which it ran has become obsolete. In addition, since this preserves each digital artifact in its original form, it must provide a way of extracting the artifact’s contents for use in future computing environments.

Two approaches have been proposed to enable running original software in the future. One is to save obsolete computers in museums where they could continue to be used; unfortunately this is highly impractical, ignores the short physical lifespan of most digital storage media, and would undermine the core digital attributes of saved artifacts by preventing access to them except at a few locations. A more attractive option is to emulate obsolete computers on future computers.

Note that all of these proposed preservation approaches – except for doing nothing – rely on saving bitstreams, assuming that the use of formal descriptions would involve digital encoding of some sort. These bitstreams must be repeatedly copied to new storage media before their current media become unreadable or obsolete. The following discussion assumes that this will be done.

Making the New Old

Emulation is the use of a new computer to “impersonate” an old computer. In other words, an emulator is a program that runs on the new computer and makes it behave like the old computer, which allows the new computer to run virtually any program that originally ran on the old computer, thereby virtually recreating the old computer. Emulation is a well-understood computer science technique that is often used to avoid the cost of adding special-purpose hardware, such as modems or co-processors, to a system or to maintain compatibility with older equipment, such as terminals or communications devices. It is typically so successful that users are completely unaware that it is being employed.

Preserving a digital artifact by means of emulation would require saving the bitstreams of all the application and system software that rendered the artifact on its original com-

puting platform, in addition to the artifact’s own bitstream. Then whenever the artifact is accessed in the future, its original software is retrieved and run under an emulator of its original computer that runs on some future computer. The emulator makes the software “think” it is running on its original computing platform, so it renders the digital artifact just as it did originally, within the limits of how well the user interface can be emulated). The emulation environment should also allow the user to extract the contents of the artifact and convert it into some future vernacular form (e.g., to quote or incorporate part of it in some new document), a process called *vernacular extraction*. This is analogous to the way ancient manuscripts and records are used: The originals are kept for scholarly use and verification purposes, but modern vernacular copies are allowed (i.e., textual transcriptions, photographs or scanned images, modern-language renditions, etc.) to be made whenever needed.

This strategy relies on emulating obsolete hardware, not software. Under this approach, its original software always renders a digital artifact, which is far superior to those approaches – or misinterpretations of this approach – that require writing future “viewer” programs to recreate the behavior of, or emulate, original rendering software. Emulation means hardware emulation.

The potential advantages of using emulation for preservation are:

- It is the only proposed approach that offers a realistic way of preserving digital artifacts in their original form.
- It allows saving digital artifacts of any format, simply by saving appropriate rendering software for that format, requiring no new software to be written for each additional format to be preserved.
- It requires no conversion of individual digital artifacts, thereby avoiding a potentially huge cost incurred by most other proposed approaches, as well as greatly reducing the chance of corruption.
- It reduces the preservation problem to one of saving bitstreams, a process that users must be able to do in any case to preserve the bitstreams of digital artifacts themselves.

The primary cost of emulation would be the creation of appropriate emulators.

Making It So

Since an emulator is a program, using it ultimately requires running it on some computer. To emulate an old computer indefinitely, the user must be able to run a program that emulates the old computer on nearly any future computer. This requires that future computers be able to perform – either in hardware or software – all the logical functions that the old computer performed. Although it may at first appear unlikely that all future computing paradigms, such as quantum computation, will subsume the capabilities of current computers, this is actually quite plausible since these capabilities are founded on simple, universal, mathematical, and logical operations whose future utility seems certain no matter what new capabilities are added to them. In any case, this is far more likely than the belief that future application programs will subsume the entire behavior of current digital formats.

If an emulator of a given computer is to be validated against the machine that it emulates, it should be implemented before that machine becomes unusable. There is normally some time during which an old platform is still usable while a new platform is gaining popularity, so it should generally be possible to write an emulator of the old platform that runs on the new platform and validate its behavior during this window. But how can this emulator be made to run on future platforms that will not come into being until long after the old platform is obsolete? Two alternative and compatible strategies for doing this are *chained emulation* and *rehosted emulation*.

Chained Emulation

Consider computer platforms of different generations, such as gen-1, gen-2. A transition to a new computer that is backward compatible with a previous computer, such as that from the Intel 486 to the Pentium, need not constitute a new generation. An emulator of a gen-1 computer is denoted that runs on a gen-J computer as em1:J. So em1:2 would emulate a gen-1 computer by running on a gen-2 computer. An emulator like this one, which spans just two generations (that is, em1:J, for J=1+1), will be called a *bi-generational* emulator.

Chained emulation involves running each bi-generational emulator under the next one (see figure at right). An example would be running em1:2 under em2:3 under em3:4 as a way of running an emulator of a gen-1 platform on a gen-4 platform. The following explains how to make this work.

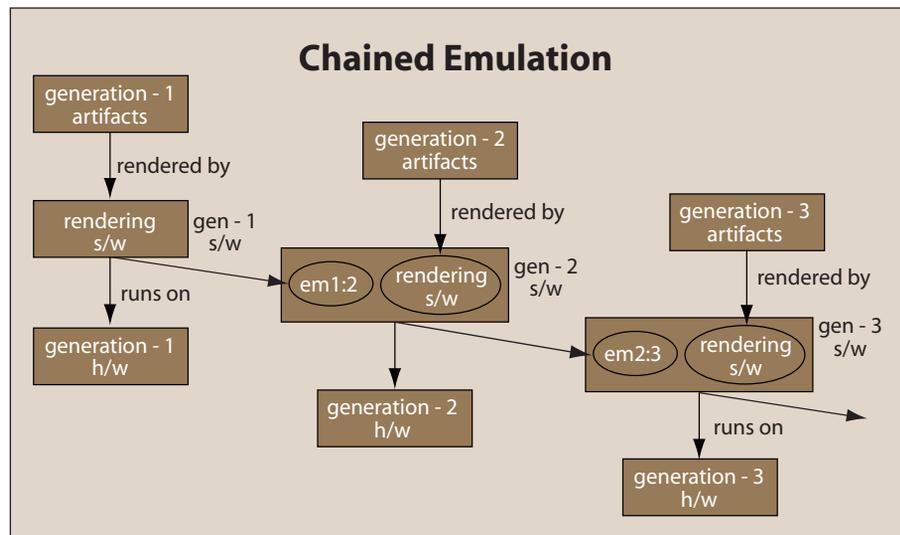
- Before gen-1 hardware becomes obsolete, write emulator em1:2 and run all gen-1 software under em1:2 in the future. (Note that em1:2 is itself gen-2 software since it runs on gen-2 hardware.)

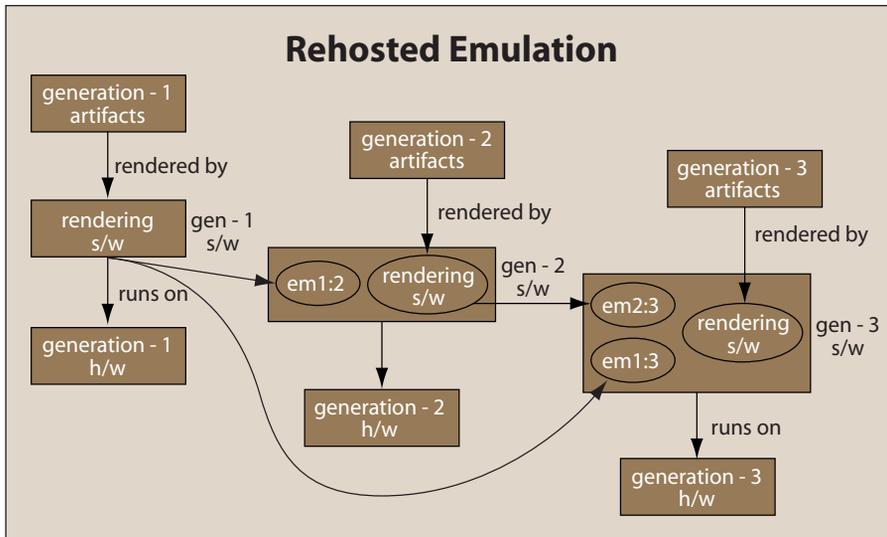
- Before gen-2 hardware becomes obsolete, write emulator em2:3 and run all gen-2 software under em2:3 in the future. Since em1:2 is gen-2 software, it too will run under em2:3, so run gen-1 software under em1:2 running under em2:3.
- Before gen-3 hardware becomes obsolete, write emulator em3:4 and in all future generations:
 - Run all gen-3 software under em3:4.
 - Run all gen-2 software under em2:3 under em3:4.
 - Run all gen-1 software under em1:2 under em2:3 under em3:4.

Under this scheme, each bi-generational emulator can be written using different programming methods and languages. Once it exists in executable, or “binary,” form, it can be considered a “black box” forever after and saved as a bit-stream. Its source code need never be re-examined or recompiled – or even saved – since it will always run as binary object code under future emulators. (Note that the vast majority of software is used as black-box object code. If the user needed to modify it or understand how it worked, it would be of little practical value.)

It may seem that chained emulation risks cumulative errors analogous to those encountered in migration, but if program P renders a given gen-1 artifact properly under em1:2 when em1:2 is first written, P should continue to work the same way so long as em1:2 can be run. An emulator of gen-i hardware captures the behavior of that hardware for all time, thereby ensuring that gen-i software will always have the virtual hardware it needs to run correctly.

Chained emulation does incur a performance penalty since many layers of emulation may need to be run in the future; however, if each hardware generation continues to be faster than the last, this should not be a serious problem.





Rehosted Emulation

Alternatively, before gen-2 becomes obsolete, the user might rehost em1:2 to run on a gen-3 computer, producing em1:3 (see figure above). Running this on the gen-3 platform would directly emulate the gen-1 platform without the need for chained emulation. It could be similarly rehosted every previous emulator on each new generation in addition to creating new bi-generational emulators, producing em1:2, em1:3, em2:3, em1:4, em2:4, em3:4, etc. To make this work:

- Before gen-1 hardware becomes obsolete, write emulator em1:2 and run all gen-1 software under em1:2 during gen-2 as when chaining.
- Before gen-2 hardware becomes obsolete, write emulator em2:3 and run all gen-2 software under em2:3 during gen-3 as when chaining; but also rehost em1:2 on gen-3 hardware, producing em1:3, and run all gen-1 software under em1:3 during gen-3. Save em1:2 for use in chaining, as a fallback.
- Before gen-3 hardware becomes obsolete, write emulator em3:4 and run all gen-3 software under em3:4 during gen-4 as when chaining; but also rehost em1:2 (or em1:3) on gen-4 hardware, producing em1:4, and rehost em2:3 on gen-4 hardware, producing em2:4. Save em1:2 and em2:3 for use in chaining, as a fallback. Then during gen-4:
 - Run all gen-3 software under em3:4
 - Run all gen-2 software under em2:4
 - Run all gen-1 software under em1:4

This is more efficient than chaining since it eliminates the need to run one emulator under another. It would, however, require creating the same new bi-generational emulator for each generation that is required for chained emulation. It would also require rehosting all previous emulators on every new generation, which involves much more work.

The two approaches, however, are not incompatible. The user must still create bi-generational emulators (em1:2, em2:3, em3:4,

etc.) for rehosting, so if all of these are saved, they can be used to perform chained emulation if it becomes too difficult or expensive to rehost all old emulators on some future generation of hardware. Rehosting can therefore be thought of as an option that can be reserved for cases when performance or efficiency is of concern.

Production Notes

Of course, each hardware generation typically consists of a number of different computer platforms, making it necessary to emulate each platform of the previous generation on at least one platform of each new generation. Even using chaining, multiple bi-generational emulators will therefore have to be written for each new generation, whereas rehosting requires many additional emulators. This suggests the need for standardized, accepted techniques to facilitate producing emulators. Two such techniques might be:

erational emulators will therefore have to be written for each new generation, whereas rehosting requires many additional emulators. This suggests the need for standardized, accepted techniques to facilitate producing emulators. Two such techniques might be:

- Write emulators in a single, standardized language that is well formalized and semantically rigorous. Also write a single interpreter or translator program for this language for each new hardware platform, thereby enabling all emulators written in this language to run on that platform. This emulator language might be a formal high-level language or a minimal, simplified language such as a subset of C (which David Holdsworth, co-author of *Emulation, Preservation and Abstraction*, refers to as "C-").
- Write all emulators to run on a virtual machine (VM) so that rehosting this VM onto future generations of hardware will allow all emulators to run on that hardware without additional rehosting. This idea is elaborated below.

A virtual machine is actually a program that runs on some host computer and makes the host impersonate a different computer. In this sense a VM is an emulator but it typically emulates an imaginary, or virtual, machine that would never have needed to exist as hardware. A well-known example is the Java VM (JVM). All Java programs are written to run under the JVM, which in turn can be implemented – hosted – on a wide variety of hardware platforms. This greatly simplifies rehosting. Instead of having to rewrite every Java program to run on every different hardware platform, the user would merely rewrite the single program that implements the JVM. Once the JVM is hosted on a given hardware platform, it can, in principle, run any Java program.

If all emulators were written to run on a single "emulation virtual machine" (EVM), then they could all be rehosted at once on a new hardware platform simply by implementing or rehosting the EVM on that platform. The Java VM is not ideally suited to being an EVM, but it would be well worth developing an

EVM. Ray Lorie at IBM, Almaden is designing a universal virtual machine (UVM) as a candidate EVM. Of course, since nothing remains constant, even a standardized EVM must be allowed to evolve over time, leading to a sequence of EVMs, beginning with EVM-1 and EVM-2. If old emulators cannot run on a new EVM, such as EVM-3, then chaining can be used to run them under emulation simply by writing an emulator of EVM-2 that runs on EVM-3 (emEVM-2:EVM-3).

Remaining Questions

Although emulation is a promising approach to preserving digital artifacts, there remain some unanswered questions about it.

Emulating peripheral devices, such as display screens or sound generators, accurately enough to retain the interface behavior of obsolete programs may require development and adoption of new ways of representing certain modalities such as color or sound fidelity. However, since most digital artifacts are intended to be usable on a range of different computers, it should rarely be necessary to emulate the specific characteristics of a single type of peripheral – let alone a particular, individual peripheral.

New techniques may be needed to represent hardware and software configurations so that emulators can run programs requiring different components. Again, since most digital artifacts are intended to be usable on a range of computers, they are often not as sensitive to such differences as they might be. Nevertheless, a general solution to this problem would be reassuring.

As previously mentioned, future emulation environments should allow users to extract vernacular versions of old digital artifacts so that they can use them in their future computing environments. Unlike migration, which converts artifacts into vernacular forms repeatedly while discarding the previous vernacular and original forms, emulation should enable vernacular extraction from the original digital form, which is rendered under emulation. Exactly how such extraction can be done remains an open issue, but current emulators of old computers that run on modern computers hint at a solution. Since such emulators are implemented using the modern computer's native environment and interface, they typically allow extracting text or images from emulated artifacts by using modern techniques such as highlighting elements of the emulated display and copying them with the mouse, after which they can be dropped into windows running modern applications or saved as modern text or image objects.

Another question that often arises is who would write emulators. One answer might be computer hardware vendors, who are uniquely qualified for the task. In any case, if organizations concerned with preservation create a market for emulators, it seems a foregone conclusion that they will emerge. Related to this question is the issue of how emulators will be tested and validated and how standards will be developed. This function might be handled by a consortia of concerned organizations or by designated national or international agencies.

Although the use of emulation to preserve digital artifacts still requires resolving these and other significant issues, its

potential low cost, universality, and ability to preserve originals – along with all of the inherently digital aspects – indicate it is well worth pursuing. ■

Jeff Rothenberg is Senior Computer Scientist at the RAND Corp. He may be contacted at Jeff_Rothenberg@acm.org.

References

Gilheany, Steve. "Preserving Information Forever and a Call for Emulators." Presented at *Digital Libraries Asia 98: The Digital Era: Implications, Challenges & Issues*, 17-20, March 1998, Singapore. Available at www.archivebuilders.com/pdf/22010v052.pdf (accessed 19 February 2002).

Granger, Stewart. "Emulation as a Digital Preservation Strategy." *D-Lib Magazine*, October 2000. Available at www.dlib.org/dlib/october00/granger/10granger.html (accessed 19 February 2002).

Holdsworth, David and Paul Wheatley. "Emulation, Preservation and Abstraction." Available at <http://129.11.152.25/CAMiLEON/dh/ep5.html> (accessed 19 February 2002).

Lorie, Raymond. "A Project on Preservation of Digital Data." *RLG DigiNews*. Vol. 5. No. 3 (15 June 2001). Available at www.rlg.org/preserv/diginews/diginews5-3.html#feature2 (accessed 19 February 2002).

Michelson, Avra and Jeff Rothenberg. "Scholarly Communication and Information Technology: Exploring the Impact of Changes in the Research Process on Archives." *The American Archivist*, 55:2 (1992). Available at www.clir.org/pubs/film/future/aapaper.html (accessed 19 February 2002).

Rothenberg, Jeff. "Ensuring the Longevity of Digital Documents." *Scientific American* 272:1 (January 1995).

—. *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation. A Report to the Council on Library & Information Resources (CLIR)*. January 1999. Available at www.clir.org/pubs/reports/rothenberg/pub77.pdf (accessed 19 February 2002).

—. *Using Emulation to Preserve Digital Documents. A Report for the Dutch Royal Library (Koninklijke Bibliotheek)*. July 2000. Available at www.konbib.nl/kb/pr/fonds/emulation/usingemulation.pdf (accessed 19 February 2002).

READ MORE ABOUT IT

De Witt, Donald, ed. *Going Digital: Strategies for Access, Preservation, and Conversion of Collections to a Digital Format*. The Haworth Press Inc.: Binghamton, NY, 1998.

Sitts, Maxine K. ed. *Handbook for Digital Projects: A Management Tool for Preservation and Access*. Northeast Document Conservation Center: Andover, MA, 2000.