Digital Preservation & Emulation: from theory to practice

Stewart Granger[*]

[*] CAMiLEON UK PROJECT Project Co-ordinator
E-mail: stewartg@dial.pipex.com

ABSTRACT
This paper considers the use of emulation for digital preservation. After describing the theoretical background provided by mathematical logic, the paper moves on to consider practical issues in the use of emulation for digital preservation.

KEYWORDS: emulation, digital preservation, Turing machines, the Church-Turing thesis.

INTRODUCTION: THE PROBLEM OF DIGITAL PRESERVATION

Many libraries and archives are in the process of 'going digital'. The advantages of digital technology are well known and its adoption by libraries and archives seems inevitable, inexorable and well-motivated. Yet the fact remains that several key issues concerning the long term preservation of digital objects remain unsolved. Most writers have tended to identify two key problems: the fragility of digital media (its 'shelf life' compared with, say, non-acidic paper is extremely short) and the rate at which computer hardware and software become obsolete. Some would argue that the first problem is not real - even if we don't know exactly how long, say, CD ROMs will last, it is likely that the drives needed to read them will be obsolete before that date arrives. Many cases have been cited in which valuable data has already been lost because of obsolescence. Digital preservation is a multi-faceted problem of which the technical issues discussed here are but one part. For an overview of some digital preservation issues see *The Task Force Report* [14] and N.Beagrie & Greenstein [12]. These problems have, of course, been exercising the library and archive communities for some time but as yet no one solution or set of solutions has been reached.

To date the most widely advocated solution to these problems has been that of migration. Migration is the process of converting a digital object that runs on one platform so that it will run on another (non-obsolete) platform. But migration has its problems and disadvantages: the process of conversion runs the risk of losing data, the 'look and feel' of a digital object and/or its functionality; it is a time consuming and therefore costly process; it is not obvious how or indeed whether all digital objects *can* be migrated (think of computer programs). For these and other reasons some believe that emulation may, sometimes at least, provide a better solution. In the CAMiLEON project we have reached the conclusion that emulation does have a valuable role to play in digital preservation [3] *Caveat*: this paper explores the possible role of emulation in digital preservation, it does not purport to explain how to write an emulator.

THE NATURE OF EMULATION

Emulation would seem to hold great promise as a method of digital preservation. This is mainly for two reasons: first, it promises a method of preservation that can preserve the functionality and the 'look and feel' of digital objects that migration may not be able to achieve; secondly it could prove to be much more cost effective solution in certain circumstances for the reason that producing *one* emulator could be much cheaper than migrating *every* digital object in an archive. Moreover, there are very strong theoretical underpinnings for emulation and I want to begin by describing some of these.

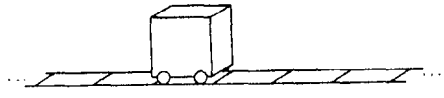
BACKGROUND THEORY

In the face of the plethora of different computers it would appear be reasonable to conclude that the idea that one machine can *emulate* another machine might appear to be at best contingent, and at worst a highly risky assumption. Yet the fact is that concepts central to the development of computer science and the mathematical theory of computation show this is not the case. Here I want only to say enough to *describe* these results.

Turing machines

Possibly the best starting place is with the concept of a *Turing machine* invented by the British mathematician Alan Turing. There are many cosmetic differences in the way Turing machines are described, mine draws on Boolos and Jeffrey [2]. The notion of a Turing machine is abstract and consists of the following:

Represented in the diagram is the notion



of a linear tape marked into squares and which is unending in both directions. With a finite number of exceptions, all squares are blank, both initially and at each stage of the computation. If a square is not blank, it has exactly one of a finite number of symbols S_1 ....S_n written in it.

At each stage of the computation, the machine is scanning one square of the tape and is capable of telling what symbol is written on the square. It is capable of erasing the symbol in the square and writing a symbol there. And it is also capable of moving one square to the right or one square to the left.

The machine is designed in such a way that at each stage of the computation it is in one of a finite number of states q_1....q_n . It is irrelevant how these states are realised physically.

The machine is controlled by a set of *instructions*. Each of these instructions is in conditional form. It tells the machine what to do depending on the symbol being scanned is S_0 (blank) or S_1 or ....S_n. There are n+4 things that can be done:

1. Halt the computation
2. Move one square to the right
3. Move one square to the left
4. Write S_0 in place of whatever is in the scanned square
5. Write S_1 in place of whatever is in the scanned square

-
-
-

(n +4) Write S_n in place of whatever is in the square.

So, depending on what instruction is being carried out and what symbol is being scanned, the machine will

perform one of these n+4 acts.
The overall program of instructions that the machine is to follow can be specified in various ways, e.g. by a machine table or by a flow graph.

What can Turing machines do?

Ignoring the issue of speed and efficiency, what can Turing machines do? To the uninitiated the answer might be expected to be 'not very much' but, the answer, amazingly, is that a Turing machine can solve any effectively solvable algorithmic problem! Harel, puts it as follows [8] :

> "....any algorithmic problem for which we can find an algorithm that can be programmed in some programming language, *any* language, running on some computer, *any* computer, even one that has not been built yet but *can* be built, and even one that will require unbounded amounts of time and memory space for ever larger inputs, is also solvable by a Turing machine. This statement is one version of the so-called **Church-Turing Thesis**, after Alonzo Church and Turing who arrived at it independently in the mid 1930s."

The Church-Turing thesis is not mathematically provable (although it is *refutable*) because it employs the imprecise concept of an 'effectively solvable algorithmic problem'. Since it appears to be such a surprising result, why should one believe it?

Evidence for the Church-Turing thesis

Harel again, [9]

> "Ever since the early 1930s researchers have suggested models for the all powerful absolute or **universal** computer. The intention was to try to capture that slippery and elusive notion of 'effective computability', namely the ability to compute mechanically. Long before digital computers were invented, Turing suggested his primitive machines and Church devised a mathematical formalism called the **lambda calculus** (...the basis for the programming language LISP). At about the same time Emil Post defined a certain kind of symbol manipulating **production system**, and Stephen Kleene defined a class of objects called **recursive functions**....All these people tried, and succeeded, in using in using their models to solve many algorithmic problems for which there were known to be 'effectively executable' algorithms. Other people have since proposed numerous different models for the absolute, universal algorithmic device....The crucial fact about all these models is that have *all* been proven equivalent in terms of the class of algorithmic problems they can solve.."

Similarly DeLong [5] writes, "It should be clearly understood that Church's thesis is an empirical thesis, not a mathematical theorem...However the evidence in its favor is so great that it should really be called Church's law since if true it is a natural law." In effect then, the most powerful super-computer can only solve the same class of problems as the simplest home computer. Non-computable (undecidable) problems are solvable on neither, and the computable (decidable) problems are solvable on both.

TOWARDS PRACTICE

It cannot be stressed too highly that the extremely strong theoretical background does not guarantee that emulation is a *practical* digital preservation strategy. Turing machines say nothing about performance or computer peripherals. In addition there is the large question of whether emulation can provide a *cost-effective* digital preservation strategy.

However esoteric most computer users may find the concept of emulation the fact is that it works. Often in fact, emulated systems work better than the original system. Some practical examples can illustrate the general point:

Example 1:

> "The 360 ...was the first machine that could emulate other computers. The smaller models could emulate the 1401 and the larger ones could emulate the 7094, so that customers could continue to run their old unmodified binary programs while converting to the 360. Some models ran 1401 programs so much faster then the 1401 itself than many customers never converted their programs." [16]

Example 2:

> A colleague who works in the CAMiLEON project has been involved with developing an emulator for the George 3 operating system which ran on ICL 1900 series computers. See [17].

Example 3:

> We often demonstrate a BBC emulator running on a PC running an early computer game ('Chukie egg'!) and early educational software.

Example 4:

> The general point is that emulation is a standard engineering technique and works for good scientific reasons.

APPROACHES TO EMULATION

The specification approach

Through a number of papers and reports Jeff Rothenberg has been perhaps the staunchest advocate of emulation as *the* solution to digital preservation problems [11] It is not the purpose of this paper to describe in detail his proposals; here I only want to mention a key feature of his approach and why this seems unsatisfactory to many people. The key aspect is the idea of developing emulator specifications – that is, a specification such that it *could* be used to develop an actual emulator when the need arose. This seems to be a risky strategy since it will be hard to know that a specification is adequate without actually developing one and testing it. One thing to be borne in mind when considering the practicalities of emulation is that one must deal with the realities of actual hardware and software. It is not unknown for software and hardware to have bugs – which, generally at least, will not be documented. Developing actual emulators is a safer way (the *only* way?) of ensuring that what one hopes will work actually works. For counter arguments to Rothenberg's approach see Bearman [1] and Granger [15].

The virtual architecture approach

IBM and the British Library are involved in a project in which the intention is to design a Universal Virtual Machine (UVM) which is then used for actual emulator development.

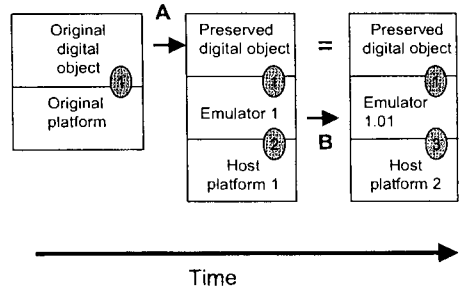
This is compatible with the approach we favour since the intention is to use it and then test the actual emulator *before* a given system becomes obsolete. Although it will be interesting to see what comes of this work, we believe this is unnecessarily complicated since a pragmatic solution is already at hand. [7]

The CAMiLEON approach

The two approaches just described so far may be said to be *generic*; that is they attempt to specify a single approach to emulation that will work for all cases. In the long run, when we are ahead in the digital preservation game this may be possible but at the moment, when we are still behind the game, this seems premature. Some work we are undertaking in the project illustrates this: we are attempting to use emulation to rescue the BBC Domesday material [6]. Here we will find it necessary to emulate a very particular video disc player and we have found that it requires research to determine the best way to produce this emulation. The practical realities of computers seem to us to preclude, at least for the moment, a generic, *a-priori*, methodology for emulation.

The CAMiLEON project is building upon work in the CEDARS project [4] which like CEDARS draws upon the Open Archival Information System (OAIS) reference model. [13] The OAIS makes a separation between preservation of digital material and the ability to provide access to its intellectual content. The CEDARS approach emphasises the importance of preserving original byte streams so that providing access to them can be achieved over time. See [10]. Clearly, a preserved byte stream is only valuable if meaningful access to its content can be provided. The CEDARS project shows the importance of identifying the significant properties of digital objects and then choosing an appropriate abstract representation of those properties that enables preservation of those properties. What those significant properties are will depend not only on the nature of the digital object but also on the purpose for which the object is being preserved. Our colleagues in Michigan are conducting some user trials to investigate what properties users find significant with respect to different kinds of digital objects.

The CAMiLEON approach has been described by my colleagues David Holdsworth and Paul Wheatley as follows [17]



The above diagram attempts to show first the transformation (arrow A) of an original digital object into a preserved digital object, which can then be run under emulation to give adequate access to the significant properties of the original. This step ensures continued access as the original platform becomes obsolete.

Further passage of time leads to the transformation shown by arrow B, in

which the obsolescence of platform 1 is handled by updating the emulator to run on a different platform. The large equals sign indicates that the preserved digital object consists of the same byte-stream in each case. The crux of the argument is that the transformations indicated by the two arrows can be implemented economically. Key to this is the design of the abstract interfaces indicated by the numbered circles on the junctions between digital components.

The interface labelled 1 is the API necessary for the successful operation of the original digital object. Emulator 1 recreates this interface in the abstract world of emulation. This interface remains the same forever. The emulator itself is a digital object and relies on interface 2 for its successful operation. A key strategic goal is future-proofing by choice of interface 2 involving only features that are likely to survive the test of time with little modification. Thus interface 3 will be very similar to interface 2, and the emulator for host 2 (emulator 1.01) will be readily derived from emulator 1.

Where emulation is concerned, there is the issue of at what level the emulation should be implemented. Choices will typically be between underlying hardware, operating system or some higher level. The best choice can, we believe, be different for different cases. In some cases emulation at the hardware level still leaves the problem of peripherals. (A problem that we believe Rothenberg considerably underestimates). In other cases it may be a drive that needs to be emulated. Factors influencing the choice here include:

- Complexity (or lack of it) in implementing the emulator
- Availability of documentation
- Mapping of peripherals into easily specified abstractions
- Retention of significant properties

The cost-effectiveness of emulation as a digital preservation strategy will in part depend upon the longevity of a developed emulator and the ease with which it may be updated when necessary. We propose several strategies for giving the source code of an emulator greater longevity:

- mulator code should be produced using standard software engineering techniques. These include the use of good code structure, informative and plentiful commenting, and good documentation.
- mulator code should be written in a subset of C, a subset chosen with a view to ensuring that only those aspects of the language semantics are used that are likely to be found in future programming languages.

Most emulators are likely to incorporate at least some non-standard code (for example for rendering of the emulated machine's display). All non-standard code should be modularised and well documented.

CONCLUDING THOUGHTS: EMULATION IN THE WIDER DIGITAL PRESERVATION CONTEXT

The seminal Task Force Report [14] on digital preservation was published in 1996. Much work in digital preservation has been undertaken since then but many of its key demands have yet to be met. The report spoke of the need for 'deep infrastructure' which included the

ideas of certified digital archives and trusted organisations. If (hopefully, *when*) these organisations are created, provided they are set up in an appropriate way (appropriate skill sets, technical capabilities etc), then they could undertake emulation work (developing emulators, maintaining emulators etc) on behalf of many user communities. Emulation, much more than migration, lends itself to a centralised cost model where a few people could do the work for many. This, potentially, has very positive implications for the thorny issue of the costs of digital preservation. It seems to me that the creation of deep infrastructure is the biggest challenge facing those of us working in digital preservation, a challenge that is largely political and not technical. [18]

REFERENCES

1. Bearman, D. "Reality and Chimeras in the Preservation of Electronic Records", *D-LIB Magazine*, April 1999.

2. Boolos, G.S. & Jeffrey, R.C., *Computability and Logic*, Cambridge University Press, 1980

3. The CAMiLEON Project:

www.si.umich.edu/CAMILEON

4. The CEDARS Project: *www.leeds.ac.uk/CEDARS*

6. Delong, H. *A Profile of Mathematical Logic*, Addison Wesley, 1971, p. 195

7. Finny, A. Domesday, at *www.atsf.co.uk/dottext/domesday.html*

8. Gilheany seems to advocate a similar approach. See "Preserving Information Forever and a Call for Emulators" available at: *http://www.archivebuilders.com/aba010.html*

8. Harel, D. *Algorithmics*, Addison Wesley, 1987, p.221

9. Harel, op.cit., p.222

10. Holdsworth and Sergeant, "A blueprint for Representation Information in the OAIS model" available at: *http://www.personal.leeds.ac.uk/~ecldh/cedars/ieee00.html*

11. Jeff Rothenberg, *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation*,1999. Available at: *http://www.clir.org/pubs/reports/reports.html*

12. N.Beagrie & D.Greenstein, *Digital Collections: A Strategic policy Framework for creating and preserving digital resources*, Arts and Humanities Data Service, 1998. *http://www.ahds.ac.uk%20/manage/framework.htm*

13. The Open Archival Reference Model, **CCSDS 650.0-R-1:** *Reference Model for an Open Archival Information System (OAIS).* Red Book. Issue 1. May 1999. Available at: *http://www.ccsds.org/red_books.html*

14. *Preserving Digital Information: Report of the Task Force on Archiving of Digital Information*, Commission on Preservation and Access and The Research Libraries Group, Inc., May 1, 1996,

http://www.rlg.org/ArchTF/

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

16. Tannenbaum, A.S. *Structured Computer Organization*, 1999, 4th Edition, Prentice Hall, p.22

17. Wheatley, P. and Holdsworth,D. "Emulation, Preservation and Abstraction" available at: *http://129.11.152.25/CAMiLEON/ep5.html*

18. I would like to thank the two reviewers of an earlier version of this paper for their helpful comments.

ABOUT THE AUTHOR

Stewart Granger is the UK project co-ordinator of the CAMiLEON project. He has over 10 years experience in the fields of digital imaging, networking and digital preservation mainly in the context of European projects in the cultural sector.
E-mail: *stewartg@dial.pipex.com*