

## OVERVIEW OF THE PROBLEM

“In a very short time, preservation has developed into a critically important part of managing a library’s most precious assets, its collections.” [46]

The vision of creating Digital Libraries rests on technological quicksand: there currently exists no viable long-term strategy to ensure that digital information will be readable in the future. The research described in this proposal attempts to fill this gap by developing what is virtually the only approach yet suggested that appears to offer a true solution to the problem of digital preservation, termed here “the emulation solution.” [43]

Documents, data, records, and informational and cultural artifacts of all kinds are rapidly being converted to digital form—if they are not created digitally to begin with. This rush to digitize is being driven by powerful incentives, including the ability to make perfect copies of digital artifacts, to publish them on a wide range of media, to distribute and disseminate them over networks, to reformat and convert them into alternate forms, to locate them, search their contents, and retrieve them, and to process them with automated and semi-automated tools. Yet the longevity of digital content is problematic for a number of complex and interrelated reasons. [1, 29, 35, 38, 42, 50]

As is now well appreciated, the physical lifetimes of digital storage media are often surprisingly short, requiring information to be “refreshed” by copying it onto new media with disturbing frequency. In addition, the technological obsolescence of these media (and of the hardware and software necessary to read them) pose an equally urgent threat. Moreover, most digital documents and artifacts require specific software to bring their bit streams to life and make them truly usable; as these programs (or the hardware/software environments on top of which they run) become obsolete, the digital documents that depend on them become unreadable—held hostage to their own encoding. This problem is particularly ironic, since digital documents can be copied perfectly, which at first glance appears to make them eternal. This contradiction prompts the cruel joke:

“Digital documents last forever—or five years, whichever comes first.” [40]

There is currently no demonstrably viable solution to this problem; yet if it is not solved, our increasingly digital heritage is in grave risk of being lost. The goal of the research proposed here is to test the viability of the emulation solution to the preservation problem [33, 34, 47].

## THE IMPORTANCE OF PRESERVATION

“The goal of any preservation program is to ensure long-term, ready access to the information resources of an institution.” [46]

Preservation constitutes one leg of a tripod that supports informational institutions such as libraries, the other legs being access and the development and management of collections [21, 45]. Without preservation, access becomes impossible, and collections decay and disintegrate.

Informational artifacts include documents, data, and records of all kinds, in all media, which we refer to as “documents” here, for simplicity [39]. The essence of preserving informational artifacts is the retention of their meaning. This requires the ability to recreate the original form and function of a document when it is accessed, for example to establish its authenticity, validity and evidential value and to allow understanding how its creator and original viewers saw it, what they were (and were not) able to infer from it, what insights it may have afforded them, and what aesthetic value it may have had for them.

Our focus is on digital documents, by which we mean informational artifacts some or all aspects of whose intended behavior or use relies on their being encoded in digital form (where the term “digital” is used in this context to denote any means of representing sequences of discrete symbolic values—each value having two or more unambiguously distinguishable states—so that they can at least in principle be accessed, manipulated, copied, stored, and transmitted by entirely mechanical means, with high reliability).

## WHAT DOES PRESERVATION MEAN IN THE DIGITAL AGE?

The preservation of digital documents involves a number of unique requirements. In particular, all such documents possess a unique collection of “core digital attributes” that must be retained, including their ability to be copied perfectly, to be accessed without geographic constraint, to be disseminated at virtually no cost, and to be “machine-readable” so that they can be accessed, searched and processed by automated mechanisms that can modify them, reformat them, and perform arbitrary computations on their contents in all phases of their creation and distribution. Furthermore, new inherently-digital (“born-digital”) document forms, such as dynamic, distributed, interactive hypertext and hypermedia, must retain their unique functionality, including their ability to integrate information from disparate sources, such as books, periodicals, news, mail, phone messages, data, imagery, and video [4, 5, 7, 27].

In response to the difficulty of saving digital documents (due to factors such as media decay and software and hardware obsolescence) it is sometimes suggested that they be printed and saved in “hardcopy” form. But this is a rear guard action that is not a true solution: printing any but the simplest, traditional documents loses much of their unique functionality (such as dynamic interaction, non-linearity, and integration), and printing any document makes it no longer truly machine readable, which in turn sacrifices its core digital attributes (perfect copying, access, distribution, etc.) Beyond this loss of functionality, printing digital documents sacrifices their original form, which may be of unique historical and contextual or evidential interest [3, 22, 52].

Alternative approaches that have been proposed (as discussed below) include translating digital documents into standard forms or extracting their content without regard to their original form. Though these approaches have traditional analogues (such as the translation of ancient texts into the vernacular to increase their audience), they are fraught with danger. The meaning of a document may be quite fragile, since meaning is in the eye of the beholder: what may be a trivial transformation to a casual reader may be a disastrous loss to a scholar, historian, or lawyer. Examples of loss of meaning abound in our daily experience of converting digital documents from their native form into that of some other software application in order to read them. At its best this often sacrifices such subtleties as format, font, footnotes, headings, numbering, shape, color, etc.; at its worst it leaves out entire segments (such as graphics and sound) or produces meaningless garbage. [24]

While it is often useful to create contemporary “vernacular” transcriptions of historical documents (such as the Shakespeare Sonnets or the Declaration of Independence), society places a high value on retaining the originals to allow us to verify that content has not been lost in transcription, as well as for scholarly and aesthetic purposes. For digital documents, retaining an “original” may not mean retaining the original medium (which rapidly decays and becomes obsolete), but it should mean retaining the functionality, look, and feel of the original document. [46]

## SCOPE OF THE PROBLEM

Preserving digital documents is more than just a basic research problem. A 1990 House of Representatives report [51] cited a number of cases of significant digital records that had already been lost or were in serious jeopardy of being lost, and the 1997 documentary film “Into the Future” [44] cited additional cases. [8, 10, 20, 31, 37]

In its short history, computer science has become inured to the fact that every new generation of software and hardware technology entails the loss of information, as documents are translated between incompatible formats [30]. Most salient are paradigm shifts (such as those between networked, hierarchical, and relational databases), which often require completely redesigning documents (or in this case, databases) to “migrate” to the new paradigm. Whenever this happens, documents that are not in continuing use may well be abandoned to save the cost of migration, while each document that migrates may be turned into something unrecognizably different from its original, which is generally lost. Even when migration does not involve major paradigm shifts, it can result in subtle losses of content and context. Yet beyond this ad hoc migration process, there exists no overall strategy for preserving (or even translating) documents over time. [9]

The scope of this problem subsumes the Digital Library domain, affecting government records, environmental and scientific baseline data, documentation of toxic waste disposal, medical records (whose lifetime must exceed 100 years), corporate data (such as drug trials documentation for pharmaceutical companies and geologic survey

information for petro-chemical companies), electronic-commerce transactions, and electronic records needed to support the forthcoming Digital Government Initiative. [8, 18]

The library and archives communities have identified at least some critical aspects of this problem and have recognized that preserving digital documents may require substantial new investments and commitments by institutions and government agencies [6, 23, 26, 48]. Showing admirable foresight, these communities have begun to discuss alternative economic and administrative policies for funding and managing digital preservation and have begun to develop conceptual frameworks for metadata that are not restricted to the print medium [15, 17, 25, 32, 41]. Yet the lack of any long-term technical solution to the problem of digital preservation limits the efficacy of such explorations: it is difficult to allocate responsibilities and assess costs for a non-existent process.

## **TECHNICAL DIMENSIONS OF THE PROBLEM**

### **Digital media suffer from physical decay and obsolescence**

There is reasonably widespread (though by no means universal) awareness of the limited physical lifetime of digital storage media. The National Media Lab has published test results [53] for a wide range of tapes, magnetic disks, CD-ROMs, etc., showing that a tape, disk, or even CD that is picked at random (i.e., without prior evaluation of the vendor or the specific batch of media) is unlikely to have a lifetime of even 5 years [28]. Vendors and media scientists often become quite emotional in arguing such numbers, but they are largely irrelevant, since physical lifetime is rarely the constraining factor for digital preservation. Even if archival quality media were introduced in the market, they would probably fail, since they would quickly be made obsolete despite their physical longevity by newer media having increased capacity, higher speed, greater convenience, and lower price. No matter how long a disk or tape may last, it is unlikely to remain readable for more than 5 years or so, because its form, size, and recording format are likely to become obsolete [45]. This is a natural outgrowth of the exponential improvement in storage density, speed, and cost. The market forces older storage media out as newer, better media become possible. Eight inch floppy disks, tape cartridges and reels of various formats, and hard-sectored disks, 7-track tapes, and many other examples reveal how quickly storage formats become inaccessible.

The dual problems of media lifetime and obsolescence have led to the nearly universal recognition that digital information must be copied to new media (“refreshed”) on a very short cycle (i.e., every few years): throughout the remainder of this proposal, this imperative will be taken for granted. Copying is a straightforward solution to these media problems, though it is not trivial. The copy process must avoid corrupting documents via compression, encryption, or changing data formats.

In addition, as media become more dense, each copy cycle aggregates many disks, tapes, etc. onto a single new medium (say CD or DVD). This raises the question of how to retain any labeling information that may have been on the original media. It is clearly infeasible to squeeze the contents of the labels of 400 floppy disks to fit on the label of a single CD. This implies that labeling information must be digitized during each copy cycle, to ensure that it will adhere to the documents it describes. But labels are directly human readable, whereas digitized information is not, so digitizing labels sacrifices one of their central functions (i.e., that they are more easily readable by humans than the contents of the media they label). This may seem a relatively trivial aspect of the problem, but it has far-reaching implications, as discussed below. [5]

### **Digital documents depend on software**

Yet the media problem is but the tip of the iceberg. Far more problematic is the fact that digital documents are in general dependent on application software to make them accessible and meaningful. Copying media correctly at best ensures that the original bit stream of a digital document will be preserved. But a stream of bits cannot be made self-explanatory, any more than hieroglyphics were self-explanatory before the discovery of the Rosetta Stone. A bit stream (like any stream of symbols) can represent anything: without knowing what is intended, it is impossible to decipher such a stream. Deciphering text is difficult enough, but a bit stream may encode not just text but also data, imagery, audio, video, animated graphics, and any other form or format, current or future, singly or combined in a hypermedia lattice of pointers whose formats themselves may be arbitrarily complex and obscure. In certain restricted cases (e.g.,

if a bit stream is known to represent simple, linear text) it may be possible to decode the bit stream without additional knowledge; but in general, a bit stream can be made intelligible only by running the software that created it or some closely related software that understands it.

In a very real sense, digital documents exist only by virtue of software that understands how to access and display them; they can be said to come into existence only by virtue of running this software.

“When all data are recorded as 0’s and 1’s, there is, essentially, no object that exists outside of the act of retrieval. The demand for access creates the “object,” that is, the act of retrieval precipitates the temporary reassembling of 0’s and 1’s into a meaningful sequence that can be decoded by software and hardware.” [46]

This implies that the only reliable way (and often the only possible way) to access the meaning and functionality of a digital document is to run its original software (either the software that created it or some closely related software that understands it) [47]. Yet such application software becomes obsolete just as fast as the digital media and media-accessing software discussed above. And although we can save obsolete software (and the operating system environment in which it runs) as just another bit stream, running it requires specific hardware, which itself becomes obsolete just as quickly. It is therefore not trivial to use a digital document’s original software to view the document in the future, on some unknown future computer (which, for example, might use quantum rather than binary states to perform its computations). This is the crux of the technical problem of preserving digital documents and is the target of the solution proposed below.

### **Additional considerations**

Any technical solution must also be able to cope with issues of corruption of information, privacy, authentication, validation, and preserving intellectual property rights. This last issue is especially complex for documents that are born digital, which therefore have no single “original” instance and for which traditional notions of “copies” may be inappropriate. Finally, any technical solution must be feasible in terms of the societal and institutional responsibilities and costs required to implement it.

## **INADEQUACY OF ALTERNATIVE PROPOSED APPROACHES**

As mentioned above, it is sometimes suggested that digital documents simply be printed and saved on hardcopy. This is not a true solution to the problem, since many such documents (especially those that are inherently digital, such as hypermedia) cannot meaningfully be printed at all or else would lose many of their uniquely digital attributes and functions if they were printed. Even digital renditions of traditional documents (such as linear text) lose their “core digital attributes” by being printed; that is, they sacrifice being directly machine-readable, which means they can no longer be copied perfectly, transmitted digitally, searched or processed by computer programs, etc. Similarly, attempting to save inherently-digital documents by printing their bit streams on paper or engraving them in metal sacrifices their machine-readability and the core digital attributes that it enables. [3, 52]

Moreover, any such scheme destroys any interactive or dynamic functionality that an inherently-digital document (or a digital rendition of a traditional document) may have, since the document’s original software can no longer be run on it. (While a human reader might be able to ignore the errors introduced by the optical scanning and character recognition of a printed version of a document, a program would be far less tolerant of the errors that this process would introduce when scanning a printed sequence of 0’s and 1’s.) For all of these reasons, saving digital documents by printing them (whether rendering their content or printing their bit streams) is not considered to offer a solution to the true problem of digital preservation.

The other approaches that have been suggested fall into three general categories: reliance on standards, reliance on computer museums, and reliance on migration. Though each of these may play a role in an ultimate solution, none of them comes close to providing a solution by itself, nor does their combination.

## Reliance on standards

On the face of it, standards appear to offer a solution by representing digital documents in forms that will endure into the future and for which future software will always provide accessibility. One version of this argument [36, 49] offers the relational database (RDB) as a paradigmatic example of how this might work. Since all relational database management systems (RDBMSs) are based on the same mathematical foundation [11] the argument goes, any RDB that is accessioned by an institution can be translated without loss into the specific form of RDB recognized by the RDBMS used by that institution. Even if the institution later changes RDBMSs, all of its RDBs should be able to migrate to the new RDBMS without loss, since all RDBMSs support the same functionality. [1, 2]

While this argument appears convincing, it fails in several significant ways, which are worth examining. First, precisely because the relational model legislates a standard baseline of functionality, real RDBMSs distinguish themselves in the marketplace by introducing proprietary features that extend the relational model (such as outer joins, support for views, unique diagrams for data modeling, etc.) Any RDB that uses such features becomes at least somewhat non-standard and will lose some of its functionality if translated into some other RDBMS [9]. This implies that standardization sows the seeds of its own destruction, forcing vendors to extend a standard in order to secure market share. Users are motivated to use such features because they provide the newest and best functionality available; but using them may create unique documents that are orphaned by standards, which enforce strict limitations on what they can represent—and thereby preserve.

Unfortunately, far from being a representative example, the relational database is actually unique: no other kind of digital document rests on an underlying formal mathematical foundation that can serve as the basis for its standardization. Though standards continue to be developed for all sorts of digital documents, these are almost always informal, ad hoc, and often short-lived. Moreover, since they lack a formal underpinning, they nearly always compete with rival standards produced for different purposes or by different industry groups. The common joke “The best thing about standards is how many different ones there are to choose from” is sadly accurate.

Finally, the relational database example demonstrates another fundamental flaw in the standards-based approach. Just as the relational paradigm replaced earlier network and hierarchical database paradigms, it is currently under attack by the new object-oriented database (OODB) paradigm, which may well replace it or at least relegate it to the role of a low-level storage mechanism hidden beneath future object-oriented database management systems (OODBMSs). As was the case with previous database paradigm shifts, the transition from relational to object-oriented cannot simply be met by automatically translating RDBs into OODBs; the paradigms are so different that such translation is typically meaningless. Even if it were possible to perform such a translation, the result would probably possess neither the formal cleanliness of the original relational form nor the semantic restructuring afforded by the new object-oriented form. This illustrates the fact that even the best standards are often bypassed and made irrelevant by the inevitable paradigm shifts that characterize information science (and are likely to continue to do so well into the future).

Proponents of standards often argue that the solution to these shortcomings is to use more standards. That is, they suggest that we force digital documents into current standard forms (even if this loses some of their functionality) and then translate them when current standards become obsolete into whatever standards supplant the obsolete ones. An analogy would be translating Homer into modern English by way of every intervening language that has existed during the past 2500 years. The fact that scholars do not do this (but instead find the earliest “original” they can, which they then translate directly into the vernacular) is indicative of the fact that translation always loses something and rarely allows us to recover the original by translating backwards again.

(Not even character encodings last forever: ASCII is slowly giving way to Unicode. Furthermore, the history of these encodings shows that successful standards do not always subsume their competitors, as exemplified by the ascendance of ASCII over the EBCDIC and APL character sets, despite the fact that it cannot represent all of their characters.)

Despite all this, standards should not be dismissed. Some standards (notably SGML and its offspring, such as XML) have proven highly extensible and worthwhile within their limited scope. Since text is always likely to be a part of many documents, SGML provides a useful capability[12], even though it does not by itself solve the problems of non-textual representation or representing dynamic, interactive documents. In fact, if SGML had been adopted as a

common inter-lingua among word processing programs, it would have greatly relieved the daily conversion problems that plague most computer users. That this has not occurred is symptomatic of the fact that carefully crafted, well-designed standards rarely sweep the marketplace [7]. Nevertheless, converting digital documents into standard forms and migrating to new standards if necessary may be a useful interim approach while a true long-term solution is being developed. And as suggested below, standards may play a useful role in a long-term solution by providing a way to keep metadata and annotations readable.

### **Reliance on computer museums**

To avoid the dual problems of corruption via translation and abandonment at paradigm shifts, some have suggested that we establish computer museums where old machines would run original software to access obsolete documents [47]. While this exudes a certain technological bravado, it is flawed in a number of fundamental ways. It is unlikely that old machines could be kept running indefinitely at any reasonable cost, and even if they were, this would limit access to old digital documents to a very few sites in the world, thereby again sacrificing many of these documents' core digital attributes.

Furthermore, this approach ignores the fact that old digital documents (and the original software needed to access them) will rarely survive on their original digital media, for reasons discussed above. If an obsolete digital document and its software survive into the future, it will probably be because their bit streams have been copied onto new media that did not exist when the document's original computer was current. For example, an old word processing file from an S-100 system will not still exist on the 8-inch floppy disk that was native to that system but will instead have migrated onto a 3.5 inch floppy, a CD-ROM, or perhaps a DVD. The obsolete document will have to be read by an obsolete machine from a new medium for which that machine has no physical drive, interface, or device software. The museum approach would therefore require building unique new device interfaces between every new medium and every obsolete computer in the museum as new storage media evolve, as well as coding device drivers for these devices, requiring programming skills for each obsolete machine. This seems hopelessly labor-intensive and ultimately infeasible.

Finally, computer chips themselves have limited physical lifetimes. Integrated circuits decay by means of mechanisms such as metal migration (where the traces that define circuit connections on the chips migrate through the substrate over time) or dopant diffusion (where the atoms that make semiconductors semiconduct diffuse away over time). Even if they were stored carefully, maintained religiously, and never used, such mechanisms would eventually render obsolete computers inoperative; using them routinely to access obsolete digital documents would undoubtedly accelerate their demise.

One role that computer museums might play in preservation is in performing "heroic efforts" to retrieve digital information from old storage media. If an old disk or tape is found that may indeed still have readable information on it, an obsolete machine in the museum (which would presumably have a drive and software for the medium in question) could be used in a last-ditch attempt to tease the bits off the medium without resorting to electron microscopy or other extreme measures. A second role might be in verifying the behavior of emulators, as discussed below.

For all of these reasons, computer museums do not offer a serious solution.

### **Reliance on migration**

The approach that most institutions are adopting (if only by default) is to expect digital documents to become unreadable or inaccessible as their original software becomes obsolete and to translate them into new forms as needed whenever this occurs [9, 16]. This is the traditional computer science "migration" approach, which, while it may be better than nothing, has little to recommend it. (It should be noted that this approach is often linked to the use of standards, though standards are not intrinsically a part of migration.)

Migration is essentially an approach based on wishful thinking. Since we can't know how things will change in the future, we cannot predict what we will have to do to keep a given digital document (or type of document) accessible and readable in the future. We can merely predict that we will often have to do something, since software and hardware

will become obsolete and paradigms will shift in unpredictable ways (unpredictability being the essence of a paradigm shift). Since paradigm shifts apply to different kinds of documents and records, we must expect such shifts to force the migration of different kinds of documents on independent, unrelated schedules, with each document type (and perhaps even individual documents or their components) requiring specialized, labor-intensive handling. We cannot predict how much effort, time, or expense will be required, how successful we will be in each case, how much we will lose in each translation, or how many of our documents will be corrupted, orphaned, or lost in each migration cycle. Furthermore, we can expect each cycle to be a unique experience with fully recurring costs, deriving little or no benefit from previous cycles, since each migration will pose a new set of unique problems.

## **CRITERIA FOR AN IDEAL SOLUTION**

In contrast to the above strategies, an ideal approach should provide a single, extensible, long-term solution that can be designed once and for all and applied uniformly, automatically and in synchrony (for example, at every future refresh cycle) to all types of documents and all media with minimal human intervention. It should provide maximum leverage, in the sense that implementing it for any document type should make it usable for all document types. It should facilitate document management (cataloging, deaccessioning, etc.) by associating human readable “labeling” information with each document. It should retain as much as desired (and feasible) of the original functionality, look, and feel of each original document while minimizing translation (to minimize both labor and the potential for loss via corruption). If translation is unavoidable (e.g., when translating labeling information) it should guarantee that it will be reversible, so that the original form can be recovered without loss. It should offer alternatives for levels of safety and quality, volume of storage, ease of access, etc. at varying costs, and it should allow changing these alternatives for a given document, document type, or corpus at any time in the future. It should provide single-step access to all documents, without requiring multiple layers of encapsulation to be stripped away to access older documents. It should offer up-front “acceptance testing” at accession time, to demonstrate that a given document will be accessible in the future. Finally, the only assumptions it should make about future computers are that they will be able to perform any computable function and (optionally) that they will be faster and/or cheaper to use than current computers.

## **THE PROPOSED SOLUTION**

In light of the foregoing analysis, we hypothesize that the best (if not the only) way to satisfy the above criteria is to somehow run a digital document’s original software. As discussed above, only by running a document’s original software can we hope to recreate its original functionality, look, and feel. The central idea of the research proposed here is to enable the emulation of obsolete systems on future, unknown systems, to allow running a digital document’s original software in the future despite its being obsolete. Though it may not be feasible to preserve every conceivable attribute of a digital document in this way, it should be possible to recreate the document’s behavior as accurately as desired—and to test this accuracy on demand.

The proposed research will address the viability of the emulation solution to the preservation problem in terms of three main research questions: (1) Can generalizable techniques be developed for specifying emulators that will run on unknown future computers and that capture all of those attributes required to recreate the behavior of current and future digital documents? (2) Can techniques be developed for saving the metadata needed to find, access, and recreate digital documents in human readable form, allowing emulation techniques to be used for preservation? (3) Can techniques be developed for encapsulating documents, their attendant metadata, software, and emulator specifications in ways that ensure their cohesion and prevent their corruption?

These questions guide the research procedures described below.

## **OVERVIEW OF PROPOSED SOLUTION**

In principle, the proposed solution involves encapsulating a number of items with each given digital document, consisting of three groups of information. In practice, there are a number of alternative ways of doing this, some of which would be safer (but more wasteful of storage) while others would involve somewhat more risk (but would use less storage). Figure 1 shows a logical view of this encapsulation. For clarity, we show all items explicitly,

corresponding to the logical model, although in practice items that are required by many different documents might be pointed to in centralized repositories rather than being replicated for each document.

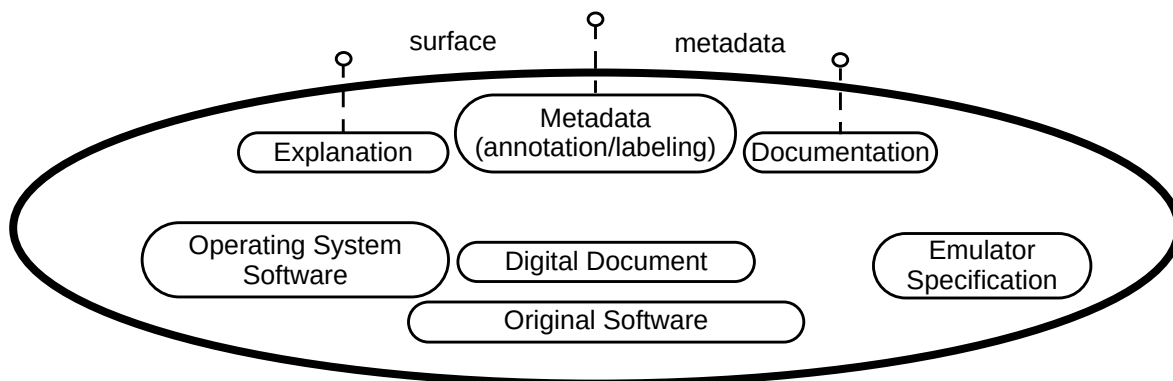


Figure 1: An encapsulated digital document

Central to the encapsulation is the digital document itself, consisting of one or more files representing the original bit stream of the document as it was stored and accessed by its original software. In addition, the encapsulation contains the original software for the document—itsself stored as one or more files representing the original executable bit stream of the application program that created or displayed the document. A third set of files represents the bit streams of the operating system and any other software or data files comprising the software environment in which the document’s original application software ran. These bit streams must be guaranteed to be copied verbatim when storage media are refreshed, to avoid corruption: ensuring this is one key aspect of the research described below. This first group of encapsulated items represents the original document in its entire software context: given a computing platform capable of emulating the document’s original hardware platform, this information should recreate the behavior of the original document.

The second group of items in the encapsulation of a document consists of a specification of an emulator of the document’s original computing platform sufficient to allow the creation of such an emulator that will run on any conceivable computer (so long as it is capable of performing any computable function). This emulator specification cannot be an executable program, since it must be created without knowledge of the future computers on which it will run: determining its exact nature is another key aspects of the research described below. Among other things, it must specify all attributes of the original hardware platform that are deemed relevant to recreating the behavior of the original document when running its original software under emulation. Note that only one emulator specification need be developed for any given hardware platform: a copy of it (or pointer to it) can then be encapsulated with every document whose software uses that platform. This second group of encapsulated items provides the key to running the software in the first group: assuming that the emulator specification is sufficient to produce a working emulator, the document can be read (i.e., accessed in its original form) by running its original software.

The final group of items in the encapsulation of a document consists of explanatory material, labeling information, annotations, metadata about the document and its history, and documentation for the software and (emulated) hardware included in the encapsulation. This material must first of all explain to someone in the future how to use the items in the encapsulation to read the encapsulated digital document. In order to serve this function, at least the top level of this explanatory material must remain human readable in the future, to serve as a “bootstrap” in the process of opening and using the encapsulation: ensuring this future readability is the final key aspect of the research described below. This is one place where standards may find a niche in this approach: simple textual “annotation standards” (which might evolve over time) would provide an ideal way of keeping explanatory material human readable. If translation of this explanatory material is required to keep it human readable (that is, if the annotation standards evolve), this can be performed when the encapsulation is copied to new media: we call this limited translation “transliteration”. Any such translation must be reversible without loss, to ensure (and allow verifying) that the explanatory material is not corrupted. (Note that these same techniques must be used to store emulator specifications,

which must also remain human readable in the future.) Additional metadata in the encapsulation describes the original document, providing the equivalent of labeling information that should adhere to the document. Finally, additional metadata must provide historical context, provenance, life-style history, and administrative information to help manage the document over time.

Given a suitable emulator specification for the desired hardware platform (which need only be created once for all documents whose software uses that platform), the process of preserving a digital document can be summarized by the sequence “Annotate, Encapsulate, Transliterate, and Emulate”. This means: (1) create any annotations needed to provide context for the document and to explain how to open the encapsulation; (2) encapsulate all of the above items with the document; (3) when necessary (optionally, at each media refresh cycle), translate annotations to keep them human readable; and (4) in the future, open the encapsulation, create the specified emulator, and run it on a future computer to run the original software under emulation, thereby recreating the document.

Though it may appear prohibitive to have to create and use an emulator to read each old document, two factors should be kept in mind. First, the inclusion of contextual annotation in the encapsulation makes it unnecessary to use emulation and run a document’s original software to perform routine management functions on the document, such as copying it, filing it, or distributing it: emulation is needed only when the document is to be read (or translated into some “vernacular” form). Second, an emulator for a given obsolete platform need only be created once for each future platform on which emulation is required to run. Once created for each new generation of computer, the emulator for a given platform can be saved to be run whenever desired. The process of generating an emulator from its specifications can therefore be relatively inefficient, so long as the emulator that is generated is reasonably efficient when it runs.

Since this approach was first outlined [33, 43], it has received considerable attention, being cited as the only proposed approach that appears to offer a true solution to the problem of digital preservation [19].

## **ANCILLARY ISSUES**

The above strategy implies some surrounding issues, which, while they are not included in the proposed research, nevertheless need to be addressed before the proposed approach can be implemented.

Saving proprietary software, hardware specifications, and documentation encounters potential intellectual property issues. Hardware specifications of the kind required for emulation are not necessarily proprietary, and since emulator specifications are not currently produced by hardware vendors (or anyone else), their intellectual ownership is as yet undefined. Application software and its documentation is often proprietary, but note that the software in question may often be merely a reader for the desired document format, rather than an editing program: such readers (along with their documentation) are often provided for free by software vendors to encourage the use of their editing software. Operating system software and drivers, on the other hand, may very well be proprietary, and intellectual property restrictions or fees for these essential items must be addressed if this approach is to work. In general, the whole point of encapsulating this software is to make it available in the future, when it would otherwise be obsolete, so one possible strategy would be to negotiate (or perhaps legislate) the free use of obsolete software.

As noted above, emulator specifications of the kind required by the proposed strategy are not currently produced by hardware vendors, or indeed anyone else on a regular basis. For this strategy to work, responsibility for developing such specifications would have to be accepted by one or more agencies, institutions, or market segments. Similarly, explanatory text standards would have to be developed and maintained, and responsibility would have to be accepted for refreshing media and performing “transliteration” when necessary (i.e., translating explanatory material into new human readable forms).

These administrative issues can be satisfied by a range of alternative policies and incentives, each with a corresponding cost model. If the proposed approach proves feasible, these alternative administrative structures need to be identified and their cost models need to be developed. Selected suites of alternatives would then need to be tried and adopted, and appropriate policies would need to be developed to implement the chosen alternatives.

## STRENGTHS AND LIMITATIONS OF THE PROPOSED RESEARCH

Note: it is not yet clear whether digital preservation needs to include the retention of attributes of the original medium on which a digital document was stored. It can be argued that digital documents are (or should be) logically independent of their storage media and therefore need not preserve the behavior of these media; however, a counterargument to this might be that since some digital documents are tailored to specific media (such as CD-ROM), they should retain at least some attributes of those media (e.g., speed). The proposal here is neutral with respect to this issue: it allows emulating attributes of storage media when desired, without requiring it in other cases.

It may appear that emulating a hardware platform simply to run application software is unnecessarily roundabout. What is really desired is to emulate the behavior of the original digital document, so why go to the trouble of running its original software at all? The answer to this (as discussed above) is that we do not yet have any formal (or even informal) way of describing the full range of behaviors possible for even the simplest of digital documents, such as are produced by word processing programs. Describing the behavior of dynamic, interactive, hypermedia documents poses a far greater challenge. The only adequate “specification” of the behavior of a digital document is implicit in its interaction with its software. This is just a restatement of the argument above, that the only way to recreate the behavior of a digital document is to run its original software.

It may then be argued that instead of actually running original software, we might emulate the behavior of that software. This would provide considerable leverage over emulating the behavior of individual documents, since a given application program may be used for thousands (or millions) of different documents, so emulating the behavior of that program would avoid having to do anything specific for each such document. However, we have no adequate way of specifying the behavior of arbitrary programs. The only meaningful “specification” of a program’s behavior is implicit in its interaction with its underlying software/hardware environment; that is, programs are “self-describing” when they run. Simply stated, the only way to tell what a program really does is to run it.

Finally, it may be argued that most application programs run under an operating system (though some may run on a “bare” machine), so we might emulate the behavior of the OS to provide a virtual “platform” for all applications that run under that OS. This would provide even greater leverage than emulating applications, since many different applications run on a given OS. (Although some applications run on several different operating systems, there are many more application programs than there are operating systems.) Emulating an OS would avoid having to emulate all the applications that run on it. However, it is at least as difficult to emulate the behavior of an OS as it is to emulate the behavior of an application program—in fact, it is probably more difficult, since an OS interacts with every aspect of the computing environment, whereas most applications are far more constrained. So the argument against emulating an application applies a fortiori against emulating an OS. Nevertheless, we do not rule out this possibility: in some cases, it may be preferable to emulate a hardware platform along with an OS to produce a “virtual hardware/software platform” that can run application programs. The approach proposed here allows for this variation, though it assumes that emulating hardware platforms will usually make the most sense.

Emulating the underlying hardware platform therefore appears to be the best approach given the current state of the art. Unlike software, we do (and must) have accurate, explicit specifications of hardware: if we did not, we could not build hardware devices in the first place. Why is it that we can specify hardware but not software? Any specification is intended for some reader or interpreter. Application software is intended to be interpreted automatically by hardware to produce an ephemeral, virtual entity (i.e., the running application), whose behavior we do not require to be fully-specified (except to the hardware that will run it), since it is intended to be used interactively by humans who can glean its behavior as they use it. On the other hand, a hardware specification is interpreted (whether by humans or software) to produce a physical entity (i.e., a computer) whose behavior must be well-specified, since we expect to use it as a building block in other hardware and software systems. Emulating hardware is therefore entirely feasible (and is in fact done routinely).

Furthermore, of the potential emulation approaches discussed above, emulating hardware has the greatest leverage. Except for special-purpose embedded processors, computers are rarely built to run a single program: there are generally many more programs than hardware platforms on which they run, even though some programs may run on multiple hardware platforms. At any given moment, there are relatively few hardware platforms in existence, though

new ones tend to appear with greater frequency than new operating systems or applications—unless we consider each version of an OS or application to be a different program, which it really is. Emulating hardware obviates having to emulate the behavior of operating systems, application programs, and individual digital documents, all of which are problematic: it is therefore by far the most efficient and viable emulation approach.

Finally, hardware emulation is a well understood, common technique. It has been used for decades both to help design new hardware and to provide upward compatibility for users, as discussed below.

## **NATURAL EXPERIMENTS SUGGEST THE FEASIBILITY OF THE PROPOSED APPROACH**

Two classes of “natural experiments” suggest that the approach proposed here will work. Though none of these solve the research questions that must be answered in order to use emulation as a basis for digital preservation, they show that key pieces of the proposed strategy have worked in the past.

The first class consists of examples of bundling digital documents with their original software to ensure that they are accessible and readable. For example, Apple Macintosh software is often distributed with a “README” file that describes the software, explains how to install it, and gives any restrictions on use, bugs, etc. (This is also an example of encapsulating explanatory annotation with software.) In order to ensure that the README file will be readable by any user, distribution disks often include a copy of a simple text editing program (“SimpleText”) that can display the README file. Though most users already have at least one copy of SimpleText on their systems, as well as other, more powerful editors, this scheme does not assume that this will be the case. The README file is bundled with software capable of reading it (i.e., SimpleText), just as digital documents are bundled with their original software in our proposed encapsulation scheme.

A second example of bundling occurs in the PC/DOS world, involving the compression scheme “PKZIP”. When a file is compressed using this software, it requires the decompression program PKUNZIP to expand the file. However, an option in PKZIP allows bundling a simple version of the unzip program with each compressed file, in this case creating an executable file which, when run, expands itself to the original file. This avoids assuming that the recipient of a compressed file will have the appropriate decompression software available: instead, it bundles that software with each compressed document.

A final example is a set of imagery distributed on the “Planetary Data Systems” CD-ROM from NASA (JPL). The imagery on this CD is designed to be read with Adobe Reader 2, a free program that can display files encoded in Adobe’s popular “pdf” format. If the user tries to run a later version of Adobe Reader to read this data, it refuses to display itself: not only does the data bundle itself with appropriate software, but it protects itself from being accessed by any other software, since it cannot guarantee that such software will treat the data appropriately. In this case, even later versions of the same program (e.g., Adobe Reader 3) are not allowed to read the data; though this may seem unnecessarily restrictive, it is in fact a safe approach, since later versions of software do not always treat files created by older versions appropriately.

All of these examples bundle software to be run on a known platform, so none of them provides much longevity for their documents; but they prove that bundling original software with a document is an effective way of making sure that document can be read.

The second class of natural experiments involves the use of emulation to add longevity to programs and their documents. The first example is the decades-old practice that hardware vendors have used to give their customers upward compatibility. Forcing users to rewrite all of their application software (and its attendant databases, documents, etc.) when switching to a new computer would make it hard for vendors to sell new machines. Many vendors (notably IBM) have therefore traditionally supplied “emulation modes” for older machines in their new machines. The IBM 360, for example, included an emulation mode for the older 7090/94 so that old programs could still be run. Apple did something similar in switching from the Motorola 68000 processor series to the PowerPC by including an emulator for 68000 code; significant pieces of the Macintosh operating system are still run under emulation. Whether emulation is provided by a special “mode” using microcode or by a separate application program, such examples prove that

emulation can be used to keep programs (and their documents) usable long after they would otherwise have become obsolete.

A second example of the use of emulation is in designing new computing platforms. Emulation has long been used as a way of refining new hardware designs, testing and evaluating them, and even beginning to write software for them before they have been built. Emulators of this kind might be a first step toward producing the emulator specifications needed for the approach proposed here: hardware vendors might be induced to turn their hardware-design emulators into products that could satisfy the proposed scheme's need for emulator specifications.

A final example of the use of emulation is in the "retro-computing" community, which delights in creating emulators for obsolete (or not-yet-obsolete) video game platforms and other old computers. There are numerous web sites listing hundreds of such free emulators that have been written to allow running old programs on modern computers. A particularly interesting example of this phenomenon is the MAME (Multiple Arcade Machine Emulator) system, which supports emulation of a large number of different platforms, suggesting that emulation can be cost-effective for a wide range of uses.

All of these examples consist of emulators that run on existing hardware platforms, so they do not address the problem of specifying an emulator for a future, unknown platform; but they prove that emulation is an effective way of running otherwise obsolete software.

## **SPECIFIC RESEARCH TASKS**

We propose research consisting of three major tasks corresponding to the three key research questions stated above: (1) develop techniques for specifying emulators that can be run on unknown, future computers; (2) develop techniques for ensuring that annotations and explanations will remain human readable in the future; and (3) develop techniques for encapsulating documents, software, emulator specifications, and associated annotations and metadata to ensure their mutual cohesion and prevent their corruption.

Each task is divided into subtasks to be undertaken in a carefully-chosen order (as discussed in the next section), to focus on the most crucial, highest-risk problems first and attempt to establish the technical feasibility of the critical aspects of the proposed solution as early as possible.

### **Task 1: Develop techniques for specifying emulators that can be run on unknown, future computers**

As noted above, the only assumption we make about future computing platforms is that they will be able to compute anything currently computable (and are likely to be faster and cheaper than current computers).

This task divides into two major subtasks: (1.1) Develop ways of specifying emulators that reproduce the behavior of emulated platforms with sufficient fidelity; and (1.2) Develop ways of saving emulator specifications so as to minimize the effort of generating emulators from them in the future. Both of these issues are crucial to the overall scheme.

Task 1.1: Specifying emulators: We will review current emulation research and commercial emulation products, hardware definition languages (HDLs), and the activities of multi-media content developers, the arcade game community, and the entertainment industry that are relevant to emulation. Based on what we learn from this, we will design an emulator specification formalism that captures all relevant attributes of a hardware platform including: interaction modes; speed (of execution, display, access, etc.); display attributes (pixel size and shape, color, dimensionality, etc.); time and calendar representations; device and peripheral characteristics; distribution and networking features; multi-user aspects; version and configuration information; etc. The formalism will be extensible so that future attributes can be added when needed (for example, for compliance with future "Y10K" standards). The set of attributes needed to ensure that a future emulation precisely reproduces an obsolete platform in all possible aspects is unbounded, but the scheme assumes that a certain degree of variance in the behavior of emulators will be acceptable. (This variance corresponds to that in the original program's behavior when executed on different contemporary systems and configurations, having different monitors, keyboards, disk drives, etc.) We will produce a

minimal list of attributes deemed sufficient to describe a range of typical current computer platforms and one or more example emulator specifications for such platforms capturing the identified attributes. By itself, this task should demonstrate that the proposed preservation scheme would work, assuming that emulator specifications can be saved so as to be used effectively in the future and that the annotation and encapsulation problems addressed in tasks 2 and 3 can be solved.

Task 1.2: Saving emulator specifications: Three main alternatives will be explored for saving specifications so that they can be used effectively to produce emulators in the future: (a) Saving an abstract, formal description that can be interpreted by a human or program in the future to enable building the desired emulator; (b) Saving an executable description (i.e., an emulator written in a high-level language) designed to run on some simple abstract machine that can be easily implemented in the future, along with instructions for implementing (and/or an abstract, formal description of) the primitives required to implement that abstract machine; (c) Making (b) an instance of (a), so that the executable description is also abstract and formal, allowing it to be used either way. We will review and analyze existing research on formal specifications, executable specifications, and abstract machine architectures and will select, adapt or develop a candidate formalism for saving emulator specifications that satisfy the needs of the emulation scheme. We will then demonstrate the use of this formalism to save one or more of the example emulator specifications produced in Task 1.1. Note that this approach requires storing formal and/or explanatory descriptions that will be used to bootstrap the process of creating a future emulator: to ensure that these descriptions will remain human readable in the future, they must be stored in the annotation form developed in Task 2.

## **Task 2: Develop techniques for saving appropriate annotations and explanations that will remain human readable in the future**

This task also divides into two subtasks: (2.1) Identify the kinds of annotations and explanations that must be saved; and (2.2) Develop ways of saving annotations so that they remain human readable.

Task 2.1: Identify the kinds of annotations and explanations that must be saved: There has been considerable work in the library, archives, scientific data, and records communities on identifying metadata to provide the historical, evidential, and administrative context for preserving digital documents (e.g., [13, 14, 25, 37, 41]). This subtask will review that work but not repeat it, focusing instead on the kinds of annotation required by the preservation scheme proposed here. We will also review current approaches to version control and configuration management and will specify the full range of annotations needed to support the emulation solution, including explanations of how to use the encapsulation itself, user documentation and version/configuration information for all the software that is to be run under emulation (and for the emulated hardware), and the emulator specification itself. We will produce example annotations for use with the emulator specifications developed in Task 1.2.

Task 2.2: Develop ways of saving annotations so that they remain human readable: Future users of digital documents preserved using the scheme proposed here will be faced with an encapsulated collection of pieces that need to be used in a particular way in order to read the desired document. At the top level, they must be able to read some intelligible explanation that tells them how to proceed. This explanation must itself be a digital document (if only so that it can be guaranteed to adhere to the other pieces of the encapsulation), but it must be human readable if it is to serve its purpose. It will generally be of the same vintage as the encapsulated digital document whose exhumation it explains, but it cannot be stored the same way as that document, or it will be equally unreadable. The solution to this conundrum lies in allowing the form of this explanatory document to be determined by the requirements of our scheme. This allows us to restrict its form, for example to simple text (or possibly text plus simple line drawings).

Even if we standardize this format, whatever standard we choose will eventually become obsolete, which is why our scheme allows annotations and explanations to be translated (“transliterated”) whenever necessary. But we must guarantee that this translation is performed without loss. This subtask therefore focuses on developing the notion of “subset-translatable” encodings, which we define as having the property that if an encoding Y is subset-translatable into an encoding Z, then anything expressed in Y can be translated into a subset Zy of Z, and anything in the resulting subset Zy can be translated back into Y without loss. This allows Z to be a proper superset of Y (i.e., not limited to Y’s expressivity) while ensuring that anything that is expressed in Y can be translated into Z and back into Y again without

loss. A sequence of such encodings, evolving as necessary over time, will solve the readability problem for annotations: each encoding in this sequence serves as an “annotation standard” during a given epoch.

Although it is logically sufficient—having asserted that an encoding Y is subset-translatable into encoding Z—to translate information from Y to Z and discard the original Y-form of that information, this is unlikely to convince skeptical future users. We therefore also propose to develop the notion of a “subset-translator” (consisting in each case of a table or a process, depending on the complexity of the translation) that shows how to translate Y into Z and back again. If this translator is saved along with definitions of encodings Y and Z and the Y and Z forms of the translated information, then any future user can verify that Y is indeed subset-translatable into Z, that the information was correctly translated from Y to Z, and that nothing was lost in this translation (by verifying that the reverse translation reproduces the original Y-form of the information). Note that to remain readable in the future, all of this saved information (encodings, translators, history of translations that have been performed, etc.) must be stored in this same manner, i.e., encoded in a current annotation standard, to be translated as needed in the future.

We will review existing research on transformation schemes and on bi-lateral translation in traditional and computational linguistics and cryptography, develop criteria for subset-translatable encodings and translators, produce examples of each of these, and delineate the requirements for annotation standards to support the emulation scheme. We will also produce at least one example of how this translation scheme can be applied retrospectively to annotations stored in an already-obsolete form.

A sequence of such translations may be needed over time, but all that is really required is to save the sequence of encodings and translators: the choice of when to translate a particular annotation is optional, so long as its encoding is not lost. This subtask will therefore also identify alternative strategies for when to perform such translations on the various items of annotation in an encapsulation and will analyze the tradeoffs among these strategies.

### **Task 3: Develop techniques for encapsulating documents, software, emulator specifications, and associated annotations and metadata to ensure their mutual cohesion and prevent their corruption**

The final piece of the puzzle required to make the proposed scheme work is to encapsulate all of the required items so that they do not become separated or corrupted and so that they can be handled as a single unit for purposes of data management, copying to new media, etc. While encapsulation is one of the core concepts of computer science, the term carries a misleading connotation of safety and permanence in the current context. An encapsulation is after all little more than a logical grouping of items. For example, whether these are stored contiguously depends on the details of the storage medium in use at any given time. The logical “shell” implied by the term has no physical reality (unless it is implemented as a hardened physical storage device). And while it is easy to “mark” certain bit streams as inviolate, it is impossible to enforce this in the face of arbitrary digital manipulation, copying, and transformation.

This task will consist of the three subtasks: (3.1) Develop ways of protecting encapsulated documents; (3.2) Explore alternative ways of achieving the logical encapsulation required by the proposed scheme; and (3.3) Develop criteria for what should be visible outside an encapsulated document.

Task 3.1: Develop ways of protecting encapsulated documents: We will review existing research on encapsulation, encryption, and authentication to identify candidate techniques for preventing logically encapsulated items from being separated or corrupted by arbitrary future processing (such as compression and encryption). To the extent that this proves impossible, we will identify techniques that can at least detect such violations or can verify that they have not occurred. Criteria will be developed for ensuring that encapsulations are not violated, and the identified techniques will be applied to the example emulator specifications and annotations developed in Tasks 1 and 2 as well as to one or more examples (to be chosen in consultation with the Library of Congress) of binary file formats and the programs required to interpret them.

Task 3.2: Explore alternative ways of achieving the logical encapsulation required by the proposed scheme: Many encapsulated digital documents from a given epoch will logically contain common items, including emulator specifications for common hardware platforms, common operating system and application code files, software and hardware documentation, and specifications of common annotation standards and their translators (as described in

Task 2). Physically copying all of these common elements into each encapsulation would be highly redundant and wasteful of storage. If trustworthy repositories for such items can be established (by libraries, archives, government agencies, commercial consortia, etc.), then each encapsulation could contain simply a pointer to the required item (or its name and identifying information, along with a list of alternative places where it might be found). We will seek advice from governmental and private libraries and archives as well as from interested commercial parties to identify issues, priorities, and perceived risks in pursuing these various alternatives, analyze the risks and advantages of relying on pointers rather than copies, and delineate a set of alternative strategies that may appeal to different institutions in different situations.

Task 3.3: Develop criteria for what should be visible outside an encapsulated document: This subtask will examine the final dimension of encapsulation, namely, what should go inside an encapsulation versus what should be presented “at its surface” to allow manipulating it effectively and efficiently. In principle, the surface of an encapsulation should present indexing and cataloging information to aid in storing and finding the encapsulated document, a description of the form and content of the encapsulated document and its associated items to allow the encapsulation to be opened, contextual and historical information to help a potential user (or a document manager) evaluate the relevance and validity of the document, and management information to help track usage, facilitate retention decisions, etc. All of this information should be readable without having to open the encapsulation, since none of it actually requires reading the encapsulated document itself.

It is logically necessary only that the tip of this iceberg protrude through the encapsulation: there must be some explanatory annotation on the surface that tells a reader how to open the encapsulation at least enough to access further explanatory information. Even this “surface annotation” will generally not be immediately human readable, if the encapsulation is stored digitally. If it happens to be stored on a physical unit of storage that is easily accessible by humans (such as a disk), then this surface annotation might be rendered as a human readable label on the physical exterior of the storage unit. But this may not be feasible: for example, if a large number of encapsulations are stored on a single unit, it may be impossible to squeeze all of their surface annotations onto the label of the unit. So in general, even this surface annotation will be on a merely logical “surface” that has no physical correlate. The immediate “reader” of this surface annotation will therefore be a program rather than a human, though it may quickly deliver what it reads to a human. One research question is how such surface annotations should be encoded, for example, whether the annotation standards developed in Task 2 are sufficient for this purpose or whether a hierarchy of such standards—corresponding to different levels of immediate human readability—should be developed.

We will explore these questions by seeking advice from data administrators, librarians, archivists, and electronic records managers and will produce an analysis of the tradeoffs involved in making more or less information available at the surface of each encapsulation.

## **PHASES OF THE PROJECT**

The first year of this project will target the highest risk aspects of the most crucial subtasks of the proposed scheme. This includes key components of subtasks 1.1, 1.2, 2.2, and 3.1 though it does not require fully completing any of these subtasks. In particular, where alternative approaches are possible (e.g., for specifying emulators), the first year’s research will attempt to identify the alternative having the lowest risk and show that it provides at least one feasible solution. Though it is unrealistic to plan specific subtasks in great detail in advance, the goals for each year of the project will be:

Year 1: Show that the overall approach is feasible, possibly assuming that certain remaining details can be resolved.

Year 2: Demonstrate a complete, feasible solution, without necessarily having explored and evaluated all alternatives.

Year 3: Produce an analysis of all identified alternatives with their advantages and disadvantages, make recommendations about the viability of the emulation solution, and identify any issues requiring further research.