

## A No-Compromises Architecture for Digital Document Preservation

Thomas A. Phelps and P.B. Watry

University of Liverpool  
Liverpool, Great Britain  
phelps@ACM.org, P.B.Watry@liverpool.ac.uk

**Abstract.** The Multivalent Document Model offers a practical, proven, no-compromises architecture for preserving digital documents of potentially any data format. We have implemented from scratch such complex and currently important formats as PDF and HTML, as well as older formats including scanned paper, UNIX manual pages, TeX DVI, and Apple II AppleWorks word processing. The architecture, stable since its definition in 1997, extends easily to additional document formats, defines a cross-format document tree data structure that fully captures semantics and layout, supports full expression of a format's often idiosyncratic concepts and behavior, enables sharing of functionality across formats thus reducing implementation effort, can introduce new functionality such as hyperlinks and annotation to older formats that cannot express them, and provides a single interface (API) across all formats. Multivalent contrasts sharply with emulation and conversion, and advances Lorie's Universal Virtual Computer with high-level architecture and extensive implementation.

### 1 Introduction

Of the many issues to digital preservation—capture (reading data from old physical media or harvesting web sites), provenance, metadata, data management, long-term storage, availability, disaster prevention, multiple data types (scientific data, video), protecting intellectual property, and others—this paper focuses on the problem of obsolescence of digital document data formats. It is an important problem: “obsolescence of media formats and data formats is the most demanding problem while preservation of bitstreams can be mastered by using well-known techniques” [17].

Within this focus of digital document formats, it is worthwhile to consider what constitutes successful preservation. For documents on paper, preservation of the physical material implies that the content can be viewed, if sometimes under restricted access. Digital works are unlike paper in that the preservation of the material itself, the data files, is trivially accomplished, once the documents have been initially collected, by successive copying.

However, viewing of digital documents is complex. Beyond a few text-based formats such as ASCII text, document formats are severely if not entirely unreadable

without decoding by specialized software. Digital documents often include time-based content such as sounds, video, and animations. Digital documents often contain active elements such as forms, scripts, and plug-ins. In the context of scientific data and software, which parallel documents with embedded programs, Messerschmitt [12] points out that the distinction between the two is “rapidly blurring” and states that “data and software preservation targets are not separate, but should be assumed from the beginning to be largely inseparable”. Marshall and Golovchinsky [9] consider the additional nuanced dimensions of literary hypertexts, “that arise from the works on-screen appearance, its interactive behavior, and the ways a readers interaction with the work is recorded”. Similar arguments could be made for research systems in general and any other document system with idiosyncratic concepts. Preservation of a viewing capability is especially problematic if the software is proprietary, the software runs only obsolete hardware, and the data formats are not public.

And viewing is not enough. More is expected of digital documents than paper. Users expect to copy and paste text, images, videos, and other content types. Institutions, web sites, and individuals all expect to search the contents of documents (this function is so fundamental it is being built into next-generation operating systems). For some users, text-to-speech and automatic Braille generation are essential. Companies and researchers want to perform text mining and automatic language translation. People want to convert documents to the format du jour, such as for handheld devices with small screens. Researchers want to infer the semantic structure of documents, utilizing all the information the document contains, everything from layout coordinates to style sheets (if any) to explicit semantic structure (if any). Users want to add hyperlinks and annotate, even if the document format does not support those concepts. We can expect the future to be increasingly demanding as new applications are invented that rely upon potentially any aspect of document content.

## **2 Related Work**

### **2.1 Hardware Preservation**

Document software, like software in general, often requires specific supporting software and, directly or indirectly (perhaps through the operating system), specific hardware. The problem is hardware breaks down, and new generation hardware may not be compatible with the software. Hardware preservation to preserve the readability of the original digital document by maintaining the original hardware and software indefinitely. Such a hardware museum is destined to ultimately fail as the hardware breaks down with no other like machine to cannibalize for parts, and parts are too specialized to resume manufacture cost effectively.

## 2.2 Emulation

Required hardware can be emulated in software on current (more powerful) computers, and therefore emulators can reproduce a document's exact appearance and behavior. It requires quite a bit of work by experts to emulate a computer, especially a modern computer, but there are many applications for such emulators and several companies sell them. When the current computer grows obsolete, a new emulator for it can run the emulator of the previous generation, and so on, creating an ever-growing stack of emulators, which may or may not be sustainable.

In any case, the document content remains trapped within the emulator. Somewhere within the emulator's memory soup are program data structures that represent "the document". But finding the document and extracting it remains at least as difficult as interpreting the document file's original bitstream. We would like to add the document content to a search engine or send the document to others to read without the overhead of the emulation stack, but cannot.

## 2.3 Conversion/Migration

Conversion, also called migration, takes material in an older format and recodes it into a newer format. This can have some success for simple data; perhaps the many formats of raster images can all be represented on a two-dimensional grid of color values. But digital documents are more complex and in general semantically incompatible from one another, and conversions from one to another almost always lose information for the fundamental reason that some concepts in one document format cannot be expressed in the other.

As a document format evolves every few years with new releases of the corresponding software, the software can usually read the last couple versions of its own format, but documents older than a mere several years may become unreadable. Thus, the conversion process requires constant attention, constant migration. This chain from format to format can lose information at every step, relentlessly degrading quality. While data loss is almost guaranteed for conversions between document formats, it is likely even within upgrades to the same software application.

Today, although emulation and conversion suffer well-known problems, they are often seen as the only ways. The UK National Archives [3] tries to mitigate the damage done by developing a database of file formats, called PRONOM, that "allows for the automatic generation of migration pathways, by identifying every possible conversion route between a source and target format, with information about how each conversion stage will affect the content". Nevertheless, even if the damage at each step is limited, when multiplied by tens or hundreds of years of conversions, and such a time span is after all the point of preservation in the first place, the data loss is substantial and certain.

CAMiLEON [10] addresses the cumulative data loss problem by always converting from the original bytestream. Documents are read into an intermediate format and various output formats can be developed as needed. The architecture was demonstrated on a selection of vector graphics formats. This is promising, but faces additional

issues when applied to more complex documents. Even among vector graphics formats, semantic gaps required elements to be downgraded, and we can expect more of this (even complete data loss in places) with complex document models, which may or may not be an acceptable compromise. It does not address document behavior, such as a JavaScript manipulation of an HTML DOM. The intermediate format seems to be a union set of concepts from all supported formats, and as a practical matter would likely become exceedingly large and unwieldy as the hundreds or thousands of document formats were adopted, many with idiosyncratic concepts and most all with innumerable small but potentially important variations on common structures.

## **2.4 Universal Format**

Some systems convert all sources into a single universal format, which it uses for all further operations. XML seems like an attractive candidate as it captures semantics and structure, is extensible, and is easy to parse. Virtual Paper [2] and UpLib [6] (neither of which claim to be a basis for digital preservation) solve the multiple format problem by capturing image and text representations of all documents, one “projection” to view and the other to search.

The most famous examples of universal formats are PostScript and PDF, which boast the unique advantage that they can already capture any document that can be printed (which is effectively all formats with static content) and increasingly more applications are generating PDF directly and at a higher semantic level than what is sent through a printer driver. In a single format, PDF supports high fidelity viewing as well as text-based operations such as searching, and the PDF file format can bundle the original document bitstream for future editing or more demanding preservation. Adobe promotes PDF for archiving [1], pointing out that PDF is a publicly available (but not open) standard and supports XML metadata records, among other features. A PDF metastandard for archiving called PDF/A [5] identifies “the set of PDF components that may be used and restrictions on the form of their use,” such as disallowing the patented LZW compression filter and requiring that all fonts be embedded.

Somehow the universal format is eternal, and perhaps it becomes so important that society ensures this. Nevertheless, the approach has its limitations. It is simply not practical to completely capture all aspects of all document formats in a union set format. The format would be unwieldy, hostile to full implementation, and would have to be updated constantly as new formats are introduced. So-called universal formats must of practical necessity select certain features and leave others behind, and thus there is a conversion step and corresponding data loss to their use.

## **2.5 Universal Virtual Computer**

Raymond Lorie proposes writing data interpreters “that can extract the data from the bit stream and return it to the caller in an understandable way, so that it may be transferred to a new system” [8]. Programs are written against a Universal Virtual Com-

puter (UVC) so that in the future, all that is needed is an implementation of the UVC on the computer of the day to run the interpreters and thus read the data.

The UVC is extremely cautious about what is certain about the future, and requires little more than the equivalent of a simple microprocessor and memory. Considering this approach from the practical point of view of software engineers charged with building a system that embraces hundreds of document formats of sometimes great complexity, this is not enough.

In practice, software engineers need an architecture outlining the large-scale organization of the software to be built and detailing the interactions among the many components. For preservation of digital documents, this architecture should embrace such domain-specific concepts as “document”, “metadata”, “text”, “behavior”, and “structure”. In practice, software engineers require a high level language, such as Java, and libraries of pre-built functions (all of which can be compiled to the UVC). A level above the UVC must interface with hardware, such as displays, keyboards, and mice.

Lorie's UVC is a solid start, and now it is time for higher-level architecture and implementation.

### 3 The Multivalent Architecture's Benefits for Preservation

We now examine the Multivalent\* Document Model to see how its architectural qualities support digital document preservation. The purpose here is not a presentation of the architecture per se (for that see [15] or a concise presentation in [16]), but an elucidation of how important aspects of the digital preservation problem are solved by certain aspects of the architecture, sometimes uniquely so.

The architecture is powerful and versatile, as can be appreciated from the following description of an earlier application of the architecture to a browser. The Multivalent Browser natively displays many document formats (PDF, HTML, scanned paper, UNIX manual pages, TeX DVI, others) and supports *in situ* annotation (highlights, notes, executable copy editor markup, Notemarks) across all formats. Annotations can attach to any point of a document (letter or image), can apply to documents that are read-only (such as the New York Times home page), anchor with Robust Locations so they can reattach correctly even if the source document has been extensively edited, and exploit Robust Hyperlinks to find a document if it moves elsewhere on the Internet.

The architecture has been implemented. The system totals over 100,000 lines and over 4 million characters of source code. The document parsers mentioned above and the browser are freely available online [13]. In the past year, the implementation has been deployed for preservation, first in the San Diego Supercomputer Center's Persistent Archive Testbed project [18].

The architecture is proven over time. Since 1997, as the API has evolved and implementation has advanced, the Multivalent architecture has remained stable. (This

---

\* Multivalent was born as a thesis project at UC Berkeley, and the creator has since moved to the University of Liverpool.

predates Lorie's UVM, but it took Lorie to indirectly point out its suitability for digital preservation.)

The architecture has many interlocking concepts, and it can be instructive to first briefly consider the totality. New document formats are supported by *media adaptors*, which are code components that translate concrete document formats into runtime data structures. The primary data structure is the *document tree*, which represents the entire content of a document (as a scroll, or a page at a time), including everything from the text and images, to scripts, to the semantic structure (hierarchy and attributes), to the physical layout. Active (programmatic) elements of a specific document or a document genre, such as hyperlinks or outline opening and collapsing, are implemented by *behaviors*, which are program code with complete access to the document contents. The particular behaviors that apply to a document or genre are listed in XML-format *hubs*.

The remainder of this section fleshes out those architectural concepts that address specific aspects of the digital document format preservation problem.

### 3.1 Media Adaptors

New document formats are supported by media adaptors, which are code components that translate concrete document data formats into runtime data structures, primarily the document tree. Currently implemented media adaptors include PDF, HTML, scanned paper of two OCR formats, UNIX manual pages, TeX DVI, ASCII, and Apple II AppleWorks word processing, among others.

Media adaptors encapsulate format-specific parsing knowledge, and are obligated to eliminate any need for further reference to the concrete bitstream. This entails correcting the format wherever needed (coercing HTML to comply to a DTD), and presenting the rest of the system with uniform word units, which may require splitting lines in ASCII or pasting together word fragments in PDF or TeX DVI.

The core system has no media adaptors officially "built in", although a few popular ones happen to be bundled with the usual distribution. The core system merely associates a MIME type or file type suffix to a hunk of code. The system provides all of the modern access to and control of documents in general. Because there is no distinction between obsolete document formats and those in current use, *obsolete document formats are as vigorous as those in current use*.

Media adaptors directly read original concrete document data formats. This avoids a problem of conversion in which bugs or approximations in one stage cumulatively degrade quality. Bugs, while always undesirable, are more benign in media adaptors, because once they are fixed, all subsequent viewings and other uses are automatically corrected. In the same way, partial implementations of formats, such as ones being painstakingly step-by-step reverse engineered, are *incrementally improvable*. Any progress can be disseminated and exploited immediately, without delaying until perfection is reached, and improvements can be distributed as they are achieved.

The capability to read original document formats is bundled with the system and therefore always *available on demand*. With conversion, perhaps the apparatus em-

ployed converted all the known documents in bulk, perhaps by a third party, and now the user encountering a new instance must revive that. With hardware preservation, the museum of hardware and software has a geographic location and even if it is on the network, it may not be amenable to opening its fragile, irreplaceable exhibits to random poking from millions of Web surfers. With a universal format, the fact that the new format may be easy to parse does us no good unless the document has been pre-processed.

Media adaptors serve as *operational definitions* of document formats. When media adaptors are the result of reverse engineering, their operational definitions also serve as the de facto specifications. When media adaptors are based on separate specification definitions, they remain essential as they illuminate the dark corners of real world (ab)use that lie outside the light of the specification. For example, HTML as found on the Web is almost never correct, and it often requires considerable correction, not to the W3C's HTML specification, but to the operation definition given by Microsoft Internet Explorer. In Multivalent, these operational definitions are part of a live system, so they are always being tuned and kept up to date.

As compared to conversion, media adaptors move the preservation problem from constant massaging of billions of documents to maintaining one media adaptor per format. Preserving individual documents is reduced to *just copying the bits*.

### 3.2 Document Tree

The primary data structure is the document tree, which represents the entire content of a document (as a single scroll, or a page at a time). The hierarchical structure of a document is directly reflected in the hierarchy of parent-child nodes in the tree, and all nodes may contain attributes. For documents such as SGML, XML and SVG, the tree directly reflects the parse tree of the document. HTML is similarly represented, but after correction to a DTD. Internal nodes of the tree are structural, and leaves hold content (text, bitmapped images). All nodes have layout bounding boxes (coordinates and dimensions), with internal nodes containing the union rectangle of their children. Ordinarily structure and layout coincide, but sometimes a special branch at the root of the tree is required to accommodate divergences such as floating images and multiple columns. Metadata is available as attributes on the root of the tree. Media adaptors can introduce new nodes types when needed, unlike HTML's Document Object Model. Remarkably, all document formats seen so far fit comfortably into a common document tree, from the fixed-format scanned paper and PDF at one end of the continuum, to the flowed HTML and UNIX manual pages at the other.

The document trees of sophisticated document formats are decorated with spans. Hypertext links and font styling are both span types. Spans provide leaf-to-leaf (and within leaf) control over appearance: font family, size, style; foreground and background color; underlining; line width; and more. Spans also control interaction, reporting keys pressed and mouse activity within the span. A handful of spans are reused by media adaptors for many different document formats.

Note that the document tree is not an intermediate format or a universal format. Unlike an intermediate format, the document tree is used directly for document appearance and behavior, preserving full document expression. (The document tree employs concepts common across formats where possible and can be used for conversion.) Whereas universal formats can bloat as the union set of incorporated formats, the document tree is tailored to an individual format at a time, free of overhead.

In support of preservation, the document tree opens *access to all document content*. Conversion, far from opening everything, typically eliminates unusual content types. Emulation hides content in an impenetrable box. In fact, in one way the document tree is superior to the original software editors/viewers for a format, because that software probably did not give access to other applications, at least not to such an extreme comprehensiveness of text, images, structure, styling, and layout.

The document tree *unifies the representation of all document formats* and lifts them to a common set of modern document abstractions. For example, operating system- and application-specific character encodings, including the various ways of dealing with large international character sets, are all normalized to Unicode. Tools and services target the abstractions and automatically work across all formats. A document analysis application has access to content, style, and layout, regardless if source was scanned paper or TeX DVI. That search engine in the previous section that adopted TeX DVI could also collect the hyperlinks on DVI to add to its crawl.

### 3.3 Behaviors

Active (programmatic) elements of a specific document or a document genre, are implemented by behaviors. A span on the document tree above is an example of a type of behavior, a media adaptor is a behavior, and outline opening and collapsing is implemented with a combination of behaviors.

Behaviors are arbitrary program code with complete access to the document contents, the network, and the disk (subject to security restrictions, but no architectural limitations). Behaviors can be arbitrarily large. Behaviors can arbitrarily edit the document tree. The sole restriction on a behavior is that it adhere to a certain interface for communication with the system and other behaviors.

For preservation, behaviors *fully embrace the active and idiosyncratic* aspects of a document format. There is no limitation to, say, what JavaScript can access of an inherently limited scripting level. For example, PDF defines a set of annotation types, such as ink and stamp, that none of the other implemented document formats do, and with a set of properties that are unlike any other document format. Each annotation type (some but not all of which are presently implemented) becomes a behavior type, and each annotation instance a behavior instance. If a literary hypertext needs a new hyperlink type with special features, it could introduce it as a behavior.

### 3.4 Hubs

The particular behaviors that apply to a document or genre are listed in XML-format hubs. Hubs use XML attributes to customize general behaviors (passing a URL to a hyperlink, for example) in the same way programming languages employ parameters for functions. Hubs use XML hierarchy to nest more complex data associated with a behavior; for example, when the user authors a note-type annotation, the note behavior is saved under the top-level, and the content of the note, its fonts and colors, and even annotations on that annotation, are nested hierarchically within the note.

Some behaviors apply only to one document (e.g., annotations), some to genres (e.g., the manual page outliner control), and others to multiple formats (e.g., a pop up menu that can send word under cursor to a definition service).

Behaviors developed for one format (or for no particular format at all) can be associated to others formats via hubs and thus bring *new ideas to old formats*. This is not strictly required by preservation (fully expressing the format is sufficient) but is necessary to bring older formats out of the ghetto and into parity with newer ones, and after all, someday today's formats will be considered ancient too. Users want to annotate all of their documents, whether PDF which has annotation types, or ASCII or WordStar (or HTML!), which do not, and hubs associate function out-of-band and therefore are free of limitations of expressiveness in these formats.

## 4 Practicalities

### 4.1 Realization

If the Multivalent Document Model defines an architecture well suited to the needs of digital document preservation, it is immediately apparent that it will require a large-scale implementation effort to fully embrace the 100s or 1000s of file formats. The large number and the fact that they are generally semantically incompatible from one another inherently force individual attention and thus demand considerable effort. But the implementation effort is no more than UVM or CAMiLEON.

Fortunately, the work is highly parallelizable to independent teams implementing media adaptors for different formats. The Multivalent architecture defines the necessary technical points of coordination, but otherwise imposes no bureaucratic overhead, and individual teams can choose formats of local importance or interest. Since all the media adaptors are part of the same architecture, common components can be shared, as is presently done for paragraph formatting of multiple fonts and for hyperlinks.

Our task is considerably lighter than the sum total of all the original document software. Preservation emphasizes appearance and behavior, not the considerable editing component of a system. Devising the document format requires considerable intellectual effort, which we merely read from a specification. We use modern technology and tools, whereas some original systems were written in assembly language to run in 48K bytes of memory.

## 4.2 Preservation of Preservation

Any preservation strategy will take maintenance to adapt it to future technologies, and our system is preserved in the same way as Lorie's UVC. In the UVC, implementations target a simple core and the maintenance problem is reduced to porting the core. Multivalent is implemented entirely in Java, and our UVC is the Java virtual machine (JVM). The JVM is directly analogous to the UVC as both are virtual machines at more or less the level of assembly language. Java's VM is somewhat more complicated, but the primary consideration is not absolute simplicity but rather a complete, rigorous definition (Java's is given in [7]) coupled with a "reasonable" level of implementation achievability. Perhaps it would not be politically auspicious for IBM to point to a Sun technology as the bedrock of its preservation strategy, but the fact that numerous companies have implemented compatible JVMs proves its viability.

It would be absurd to build a large system in the assembly language of virtual machines. Moving from satisfaction of the key self-preservation requirement to the software engineering considerations of building a large system, we must choose a high-level language. (The use of Java's VM does not imply the use of the Java language itself, as many programming languages can compile to the VM, just as many programs can compile to the different microprocessors. Different groups could choose different languages and effectively cooperate.)

## 5 Future

The Multivalent architecture is well suited for preservation, but was originally designed for use in a browser. It could benefit from the insights of experts in preservation to ensure that the overall approach fully embraces all essential details before a large-scale implementation effort is launched.

An important subproject will be the collection of document format specifications. These are important for their intrinsic status as the intentional definitions, and considering how time consuming reverse engineering is, it is important for software engineers to have easy access to these specifications. Wheatly [19] catalogs numerous books, web sites (for example, [20]) and projects that collect many types of file formats. Companion subprojects should collect implementations, which are the operational definitions of formats, and sample documents for developers.

Undoubtedly the present technology will need to be generalized and refined, and already one area in particular area is evident. Documents are often found in wrappers of various kinds, sometimes for compression (such as .zip files) and sometimes in virtual filesystems (such as the Structured Storage used by Microsoft Office applications). A layer underneath document parsers would need parse these structures and provide access to the documents inside.

This paper has concentrated on digital documents, but there are many other media types (some of which are embedded in documents) in need of preservation, such as scientific data, audio, music scores, video, multimedia such as Macromedia Flash, and DVD menu programs, to name a few. It is unclear whether these all can be accommo-

dated under a common architecture. But Multivalent has already demonstrated its applicability to a variety of documents with text, images, vector graphics, and programmatic manipulation of a document tree. Even it is limited to this class of a couple hundred formats, billions of document instances make a claim for some social value.

## 6 Conclusion

Compared to existing approaches to digital document preservation, the Multivalent Document Model offers a step forward. Compared to conversion, the original document remains perfectly preserved. Compared to emulation, the content of the document is easily available. Compared to Lorie's UVM, Multivalent defines the high level architecture necessary for software engineers, and Multivalent's implementation of number of complex and obsolete document formats prove the architecture's power and no-compromises suitability for preservation. Multivalent is a proven plan in the present for the future of preserving the past.

**Table 1.** Comparison of selected systems used for digital preservation

	PDF	UVM	CAMiLEON	Multivalent
Defined	1990	2001	2001 (?)	1997 / applied to preservation in 2004
Demonstrations	<b>everything</b> that can be printed	JPEG and GIF bit-mapped images (claimed PDF in fact based on a conversion to HTML)	interconversion among SVG, Draw, WMF vector graphics	<b>PDF, HTML, scanned paper, TeX DVI, UNIX manual pages, Apple II AppleWorks</b>
Method	printer driver captures print stream, or app directly generates. Format eternally supported.	read original bit-stream by document interpreter	read original bit-stream into intermediate representation, convert to another file format	read original bit-stream and build runtime data structures
Strengths	captures static aspects of all formats, well developed, well defined	potential to <b>fully express</b> document appearance and behavior	as compared to other conversion, only one level of quality degradation	<b>fully expresses</b> document appearance and behavior
Use by Applications	use Acrobat or third-party library	undefined	App du jour picks up output file format	Live runtime linking (also amenable to conversions)
Document Architecture	emphasis on graphical appearance; structure expressible but not common	undefined	intermediate representation (either unwieldy union set of all formats, or leave out idiosyncratic)	<b>fully developed</b> (media adaptors, document tree with structure and layout, behaviors, spans, hubs, ...; fixed and flowed layouts)
Software Engineering	File format well documented, Acrobat API, many third-party libraries	low-level assembly language UVM (in practice use Java)	(unknown)	well-exercised system API, high-level language (Java)
Maintenance	Upgrade money to Adobe	port UVM to new machines	develop new output formats, "software longevity principles"	port Java VM to new machines
Drawbacks	everything must look like PDF (fixed layout, paginated): lose idiosyncrasies and behavior)	Implementation immature: No document architecture. UVM too low level for development	Conversion's semantic gap between formats downgrades or loses data. Loses behavior.	Intimate linking by apps, or develop own apps. (No compromises for document quality.)

## References

1. Adobe Systems, Inc. *PDF as a Standard for Archiving*, Adobe white paper. <http://www.adobe.com/products/acrobat/pdfs/pdfarchiving.pdf>
2. Birrell, A. and McJones, P. Virtual Paper Web site, (1995–1997). <http://www.research.compaq.com/SRC/virtualpaper/>
3. Brown, A. Preserving the Digital Heritage: Building a Digital Archive for UK Government Records, In *Proceedings of Online Information*. (2003) <http://www.nationalarchives.gov.uk/preservation/digitalarchive/>
4. IBM. Digital Asset Preservation Tool Web site, <http://www.alphaworks.ibm.com/tech/uvc?Open&ca=daw-fiHts-120204>
5. International Standards Organization. ISO/CD 19005-1, *Document management—Electronic document file format for long-term preservation—Part 1: Use of PDF (PDF/A)*, November 31, (2003). [http://www.aiim.org/documents/standards/ISO\\_19005-1\\_\(E\).doc](http://www.aiim.org/documents/standards/ISO_19005-1_(E).doc)
6. Janssen, W.C. and Popat, K. UpLib: a universal personal digital library system, In *Proceedings of the ACM symposium on Document Engineering*, (2003) 234–242
7. Lindholm, T. and Yellin, F. *The Java Virtual Machine Specification, 2nd edition*, Addison-Wesley Longman Publishing Co., Inc. (1999)
8. Lorie, R. Long term preservation of digital information, In *Proceedings of the First ACM/IEEE-CS Joint Conference on Digital Libraries* (2001) 346–352
9. Marshall, C.C. and Golovchinsky, G. Saving Private Hypertext: Requirements and pragmatic dimensions for preservation. In *Proceedings of ACM Hypertext* (2004) 130–138
10. Mellor, P, Wheatley, P, Sergeant, D. "Migration on Request : A Practical Technique for Digital Preservation" ECDL (2002)
11. Meehan, J., Taft, E., Chernicoff, S., Rose, C., Karr, R. *PDF Reference, fifth edition*, (2004)
12. Messerschmitt, D.G. Opportunities for Research Libraries in the NSF Cyberinfrastructure Program, *ARL Bimonthly Report* 229 (2003). <http://www.arl.org/newsltr/229/cyber.html>
13. Multivalent Web site. <http://multivalent.sourceforge.net>
14. The National Archives. PRONOM Web site. <http://www.nationalarchives.gov.uk/pronom/>
15. Phelps, T.A. *Multivalent Documents: Anytime, Anywhere, Any Type, Every Way User-Improvable Digital Documents and Systems*, Ph.D. Dissertation, University of California, Berkeley (1998)
16. Phelps, T.A. and Wilensky, R. The Multivalent Browser: a platform for new ideas, In *Proceedings of Document Engineering*, (2001) 58–67
17. Rodig, P., Borghoff, U.M, Scheffczyk, J., and Schmitz, L. Preservation of digital publications: An OASIS extension and implementation, In *Proceedings of the ACM Symposium on Document Engineering*, (2003) 131–139.
18. San Diego Supercomputer Center. Persistent Archive Testbed (PAT). <http://www.sdsc.edu/PAT/>
19. Wheatley, P. Survey and Assessment of Sources of Information on File Formats and Software Documentation (2003) [http://www.jisc.ac.uk/uploaded\\_documents/FileFormatsreport.pdf](http://www.jisc.ac.uk/uploaded_documents/FileFormatsreport.pdf)
20. Wotzits Format? Web site. <http://www.wotsit.org/>