On the duration of rich documents

Fabio Vitali
University of Bologna

Abstract

The issue of preserving electronic document is extremely important, especially since we are witnessing the transition from a paper-based publication of information to a computer-based one. In the past, many naiveties have caused enormous problems in the correct preservation of electronic documents. Hopefully, some methods have been devised now to prevent new problems to occur for current electronic data. One such method is the reliance on standard, text-based, data-oriented data formats, among which particularly important is the Extensible Markup Language, a generic data meta-format that is rapidly gaining support. Unfortunately, though, the rise and diffusion of virtual documents makes preservation of even the best structured document impossible, unless the preservation is activated at the source, by the original owner of the information. Projects of archiving the whole Internet in spite of the willingness of the data owners are intrinsically doomed.

Introduction

> "When you reach the top of the stairs you are in a huge room several stories tall. It is dimly lit from a slot cut through the living rock of the mountain on the southern face. […] You are able to show the dial around [a] sphere, now showing you the year in the cryptic method of keeping time when this clock was built. It reads the year 11.567. […] You are struck that the people of this ancient time had the foresight to think this far into their future and create this place.
>
> At this point you wonder through the rest of the facility to find a library and people accessing and preserving the data stored there. Akin to the truly ancient library of Alexandria, there is a constant forward migration of the data to increasingly better and denser methods of storage. In the main vault you find the original 1.000 books stored at the impossibly large scale of 100 nanometer pixels. These were the first 1.000 books stored in the Clock/Library chosen by its founders. Although not necessarily relevant to your time, what they began helped to teach people the value of knowledge over long periods of time. Without it humanity might have obsolesced itself out of existence without being able to look over the ancient records of the sea and air and find trends that are only apparent over centuries or millennia" [1]

A gigantic, fascinating, heroic engineering project is being designed and organized in these years straddling between two centuries, two millennia, possibly two civilizations: the long-term project to build a large mechanical clock, powered by seasonal temperature changes, in order to provide fuel and meaning to all thoughts overcoming the pathologically short attention span that imbues our times. As the computer designer Daniel Hillis proposed in 1995:

> "When I was a child, people used to talk about what would happen by the year 2000. Now, thirty years later, they still talk about what will happen by the year 2000. The future has been shrinking by one year per year for my entire life. I think it is time for us to start a long-term project that gets people thinking past the mental barrier of the Millennium. I would like to propose a large (think Stonehenge) mechanical clock, powered by seasonal temperature changes. It ticks once a year, bongs every once a century, and the cuckoo comes out every millennium."

The project is symbolic in its conception, shape, and intended effect: a stable, reliable, long term construction of a big and impressive slow clock, aimed at inducing thoughts of long term projects, endeavors, responsibilities. Along with the Clock, a huge Library will be built, to store permanently (or as permanently as we can now imagine) and incrementally all the texts and data that have long term relevance, significance, effect. The purpose is to dilate the

concept we have at this moment of the Now, which has shrunk enormously in just a few generation, and put our current action in a right perspective with the effect in the future.
The *Clock of the Long Now*; as the project is called, and the Library associated to it, constitute one important signal of the growing interest in the long term management of our culture. Major and influential personalities are part of the steering committee, such as Esther Dyson, Paul Saffo, Mitchell Kapor, etc. More information about the project can be found in the relative Web site, http://www.longnow.org/, and in [1].
How would one design such library? Would it be digital or printed? How much information should be put in it? What is the physical medium that can presumably store that quantity of information over its foreseen life span of ten thousand years? Printed with ink on acid-free paper? Or stored on magnetic tape? What data format can we use, that could last ten thousand years? Roman letters drawn with black ink on acid-free paper? ASCII text? HTML? Microsoft Word?
Even information to last a century would constitute at the moment an unreachable goal. We can hardly make paper leaves that last a hundred year, or digital media that last a few dozens years. The digital world allows, indeed, to rapidly and safely copy information from old media onto new media, but this requires constant awareness, constant attention and constant care, for digital media, when it fails, fails utterly and definitely. A loss of a few bytes in critical positions of a disk (in the boot sector, perhaps) may render several gigabytes of data completely unreachable.
But even shorter, it seems, is the life span of data formats: although we can obtain a perfect copy, byte by byte, of a WordStar or a WritePerfect document that was written fifteen years ago, we will most probably not be able to open it with any word processor, although those applications were among the best selling applications of the time. The problem worsens for graphic formats, and even more for custom-built applications.
The real problem with data formats is that for several years the persistent representation of data on disks has been considered a byproduct of the activation of the corresponding computer programs: document files were meant to be created with the same application that would later be used to display them, modify them, print them. Since applications often get updated, go out of fashion, are not ported to new hardware architectures or new operating systems, accessing legacy files soon becomes a problem. The first years after the demise of a popular application, most new applications will probably include a converter from the old format to the new one. But rapidly these options are felt useless, unnecessary, hard to maintain by the programmers, and are soon discarded from subsequent releases.
If a file format depends on the application that can write and read it, then the life expectancy of the file format is identical to that of the application itself. But if the data format is not strictly connected with the application, then the two life spans become independent, and we can hope to get subsequent, innovative applications that can and do read and write the same data format.
Standards are good for this: new applications appear every month, but good standards take years in creating, and have much longer life expectancies. GIF and JPEG, both formats for graphics and images, were not designed to work with any specific application, but several generations of independently created applications, over several years, have been programmed to handle pictures stored in these formats.
The XML language, like its predecessors SGML, was created for documents that have to last. The emphasis on describing the semantics of the data, rather than the functionality that needs to be activated with it, makes SGML and XML documents independent of these functionalities, and flexible to undergo both old and new applications without changes.
But even though obsolescing data formats may be won by standard and semantically rich data formats, the problems of preserving data is more complicated still. Virtual documents created dynamically by complex applications on the Internet make the very object of preservation a

fuzzy and evanescent concept. It is unclear whether we can and should preserve any single output of these applications, or we should try to preserve the very engine that can create these output.

What of ongoing drafts, ever-changing documents that never reach a final, preservable form? What of chats, portals, weblogs, and all those Internet live documents whose form really can have neither a printed form nor a content persisting in time?

As usual, solving a problem changes the very problem. Electronic information have been difficult to preserve, to migrate to newer hardware and software architectures, to access even after a few years after the time it was created. The Internet and the World Wide Web have brought forth the idea of a network of everlasting documents that are easy to access, to mirror, to backup and restore. Yet, the shameful confusion created by abusing HTML, by forcing in our Web documents those meager graphical effects that HTML could provide, does already shorten considerably the useful life of our documents. Even worse: dynamic, virtual documents can not have, for specific and intrinsic reasons, a preservable state. Thus, how can we preserve them?

But first things first. We had better start by examining the issue of long-term preservation of computer data, and determine if there are classes of documents that will be intrinsically difficult to preserve. Then we should concentrate on XML as a specific solution for the preservation of a large class of document types, besides being a very good idea for many other aspects of document management. Only finally we will contemplate the preservation problems of dynamic and virtual documents on the Internet.

What are we preserving, anyway?

Consider this paper, a simple computer file created with one of the most popular word processing applications currently on the market on a rather popular operating system. In order to know how to preserve it we need to understand how the paper ends up in bits, currents and magnetic fields on the media we choose to use.

The words that make up this paper are stored according to a specific *character encoding*, that transforms the letter of my alphabet into bit patterns. The word processing application decides the encoding based on the operating system and the human language of the document.

The word processing application provides numerous functionalities for formatting, that would allow me, were I interested, to paginate the paper with sophisticated layout and typographical effects. In reality, I don’t really care for these formatting, since the paper will be handled by a professional editor that will override the formatting instructions that I might have selected and will provide a consistent appearance and style to all the papers of the same publication. Still, the word processing adds information about the –albeit simple- formatting options that I have selected and stores it alongside with the words making up the paper. Thus the actual file is composed of both my words, and the commands of the word processing application. The designers of the program decided that these two types of information would be mixed up in order to maximize the efficiency of the application, and thus what the program writes on disk is a complex and unreadable mass of characters within which my words are difficult, if not impossible, to sort out.

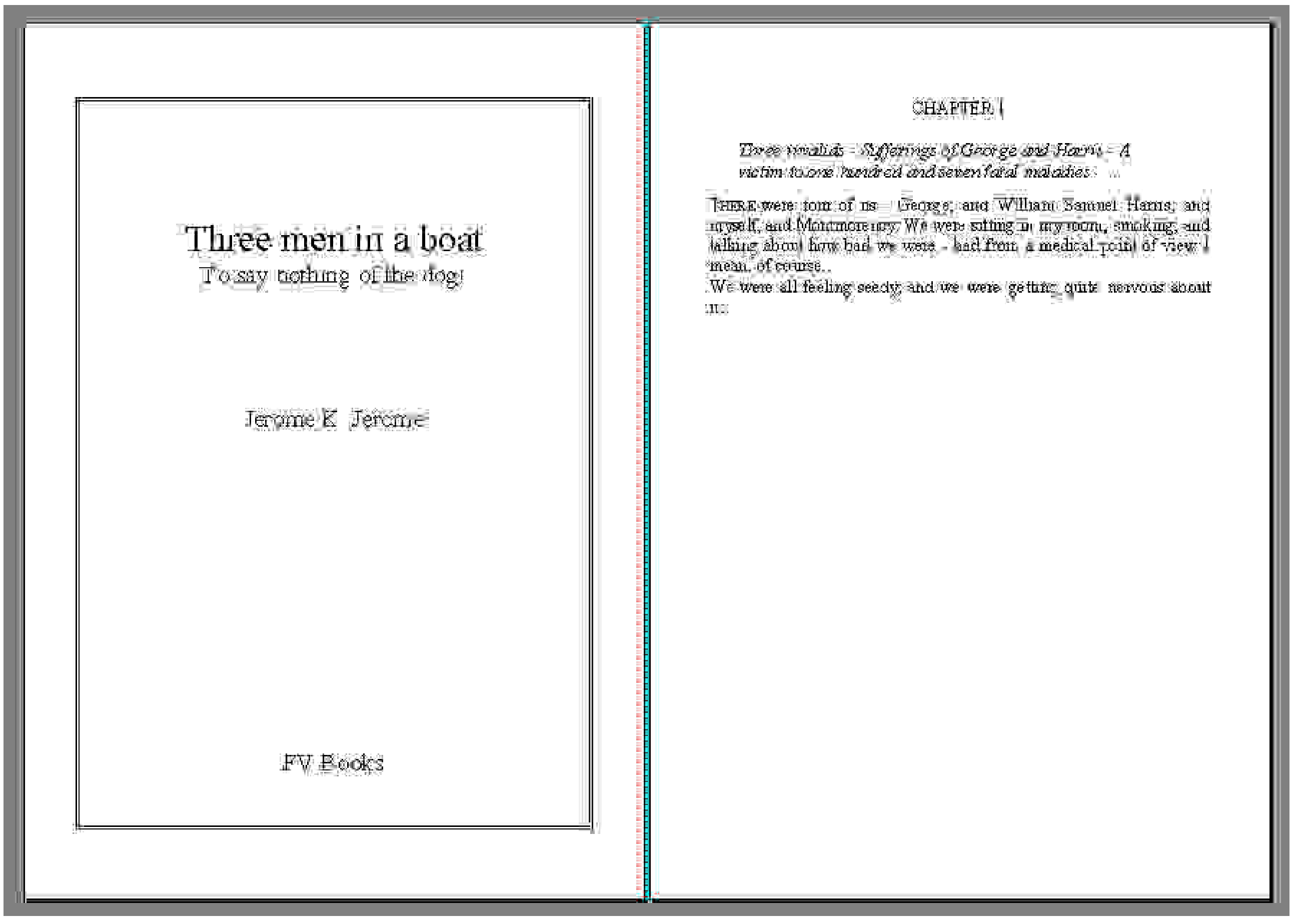
The application does not directly access the physical media on which the document will eventually be stored, but instructs the operating system to do so for it. The operating system receives the document as a block of data, and, completely ignoring the content, encoding and format of the document, it wraps it with some additional information useful for cataloguing and retrieval: a user-decided name, an access path, the current date and time, the name or identifier of the application itself. Lower levels of the operating system, at the file system level, may well decide to cut the data in fixed-size chunks that optimize the storage on the media, and provide means to reconstruct the original data from all the pieces.

The operating system, though, does not directly control the storage medium, but relies on an additional piece of software for this, the driver of the peripheral. This driver knows the details of the unit it is controlling, what commands it can perform, in what order the commands should be given, and what to do in case of one of several dozens of error types.
The unit finally receives the commands from the driver, and performs the actions requested, transforming the data different it receives in minuscule magnetic fields, in electric currents of minuscule electronic circuits, in reflection properties of minuscule areas of the media surface, etc. The document is finally stored in a persistent manner on a physical medium
As we have seen, there are several independent layers of software and hardware activities between my words and my disk:

- Character encodings (to convert letters of an alphabet in a predetermined bit pattern)
- Data formats (mixing content – my words – and application functions – formatting properties of the document – in the same data entity)
- Operating systems (providing information about my data – metainformation – and splitting parameters for archival and retrieval efficiency)
- Hardware drivers (providing support for the actual operations of the unit)
- Hardware units (performing the operations intended to persistently activate the medium to store the data)

In order to design the preservation of documents such as a paper, we have to consider all these aspects in all their ramifications.
The most evident characteristic of these aspects is their volatility. Continuously and frequently, all these aspects evolve, change and outdate older choices: new applications, or new versions of the same application, are made available that adds different types of data to the content, new operating systems handle files and metainformation differently, new drivers need to be loaded to exploit newer functions of our old and new hardware, and even the character encoding is not stable, but new encodings are coming out to provide support for more alphabets, for documents in multiple languages, etc.
Usually, of course, new software and hardware provide support, or at least upgrade paths, for older generations of themselves, but rare is the case where this support spans backward for more than one or two generations: a distraction of a few years, and already some data cannot be read with newer computing setups.
Fortunately, with a minimum of care, we can overcome most of the problems provided by evolving hardware, drivers, and operating systems; as noted in [2], there are now ways to handle unavailability of hardware and software platforms. *Migration of data* on newer media or operating systems provides a longer life to our data. Fortunately, copying comes at a continuously lower cost, since newer media usually cost orders of magnitude less than older ones, and computers are orders of magnitude faster that those of the previous generations. Thus we can easily estimate that, even though we will keep on accumulating more and more data on devices that in time will become obsoleted, it will become easier and cheaper to copy it on newer devices.
Furthermore, the availability of large, inexpensive hard disks and wide area networks may further improve our future in this regard. The habit of using tapes for long-term storage of copious quantities of data is disappearing, and huge databases of public use are now frequently put on line, on hard disks of networked computers. While our recent past is filled with situations whereby old, off-line tapes become unreadable with time (as noted in [2] again, the 1960 American Census data were recovered incompletely and with incredible effort, and the 1970 satellite observations of the Amazon basin are now completely lost), we can safely assume that now migration of data, or at least network backups, are becoming a routine task whenever newer hardware is being installed. It is well known that disk space is growing at a greater speed that our capacity of filling it [3], and analogously for the time needed to copy it.

Nonetheless, these somewhat reassuring thoughts are not possible for the more evanescent aspect of obsolescence: data formats. What will we do if our perfectly preserved huge quantities of data can only be read by an application that has gone out of fashion years ago, and is now incompatible with the current generation of operating systems?
Sometimes, the applications themselves are what is deemed worthy of preservation. For instance, video games [4] have peculiar characteristics with respect to other types of applications: games are usually extremely dependent on the hardware architecture they were designed for, and thus badly migrate to newer systems, even when these claim full compatibility with old ones. Furthermore, games are usually not important for the documents they can produce, but for themselves, the activity that they perform, the entertainment value that they, by themselves, give to their users. The best way to preserve these artifacts is through the emulation of the hardware platform they used to run on. The retrogaming community therefore has chosen to produce emulators for making obsolete games available on current generation computers. For instance, the MAME project [5] aims at producing emulators for most of the arcade game computers produced in the eighties. Similar projects exist for emulating old home and game computers, such as the Commodore 64, the Apple II, the Nintendo Gameboy, the Spectrum ZX, etc.
Most of the times, anyway, it is not applications that we want to preserve, but the documents we have produced with them. Preservation or emulation of the original hardware platform on which the applications used to run is a sub-optimal solution: as Steve De Rose notes [6], "why would I want software and data doomed forever to run only on the machine it started on? Typically, it would run much better - not just faster - on newer devices. In many cases, the original target hardware is barely capable of running software anyway, given programs' tendency to expand to fill the space and power available to them. [...] If one encapsulates everything, new technologies cannot be applied. What if I want to use the data in a new way? A slavish adherence to the limitations of old media is often passed off as if it were the virtue of preserving the strengths of those same old media. [...] Thus systems often slavishly emulate [old media's] weaknesses, or even more commonly forget to support the electronic medium's compensating strengths."
As we see it , the preservation of data is not simply a matter of survival of computer bytes across time, but rather it is a way to rephrase all the organizational aspects of the production of these bytes.



Three men in a boat
To say nothing of the dog!

Jerome K. Jerome

FV Books

CHAPTER I

Three invalids - Sufferings of George and Harris - A victim to one hundred and seven fatal maladies

THERE were four of us - George, and William Samuel Harris, and myself, and Montmorency. We were sitting in my room, smoking, and talking about how bad we were - bad from a medical point of view I mean, of course.

We were all feeling seedy, and we were getting quite nervous about it.

Fig. 1: a very simple MS Word example

We have to understand that a computer document is not important just for the tasks we perceive important when we create it, but that in time new uses may arise for these data that we could not foresee when we created them. If we organize the production of our data so as to allow new, unforeseen uses of them, we can not just preserve our documents, but extend their useful lifetime.

The data format is a key factor in this organizational restructuring. Leaving the decisions on the data format to the application designers is simply unacceptable: the many economical and technical constraints would lead them to just choose the data format that is easiest to implement. These data formats may have three main problems:

· They may be **binary**, that is, a plain dump of the data as it is stored in memory by the application. In binary data there is no discernible structure in the data, and only the original application has the necessary knowledge to determine, by virtue of position alone, the role and meaning of each chunk of data. This makes the data format hard to interpret in absence of the original application, and the data itself provides no hint of how it should be read.

· They may be **proprietary**, that is, a format whose details are decided by the application designers only. These formats are often not given to the public, and even if they are, they are usually subject in any moment to any modification the owners decide to do. Only the owners have a solid chance, and often the commercial interest, to provide software that reads and writes this format.

· They are **application-oriented**, that is, they contain direct references to specific functions every applications provide a set of functions that can be executed by users to manipulate their data. Usually these functions add information to the data: they may specify that some text is to be shown in bold, or that some part of an image has a special color palette, etc.

The added information, which we will call markup, usually reflect directly the functions of the application, and cannot be preserved without the original application.

Most application data formats are binary, proprietary and application-oriented. In order to clarify the properties of these kinds of format, I will provide here a simple example, using MS Word as the source application. In fact, MS Word can write documents in a variety of formats having many of the above mentioned characteristics.

The simple document we will consider is the beginning of a novel as could be prepared by a publishing house. In fig. 1 I show the result on screen using the MS Word application.

MS Word allows its documents to be saved in several formats. The standard format is a binary format (the so-called ".doc" format), which is shown in fig. 2.

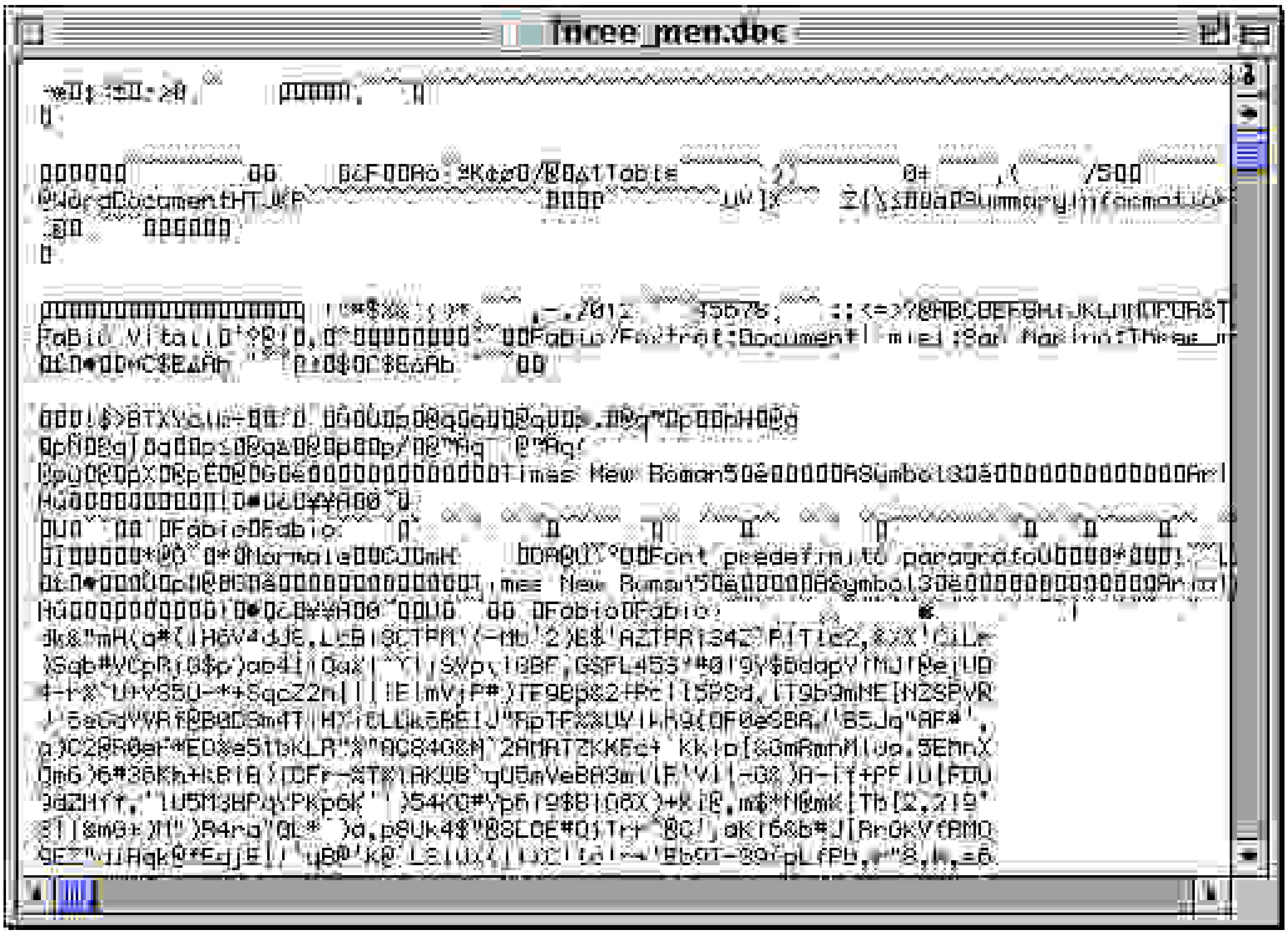


Fig. 2: the binary format of a MS Word document

There is little of immediately understandable in the binary format of Word: the document as stored on disk contains most of the information elements in a manner that can only be interpreted by an application. Some of the text can be discerned, but it is impossible to understand, at a glance, the role and meaning of the majority of the content of the file.

MS Word also provides the possibility to save the document in an equivalent, non-binary format, called RTF (Rich Text Format). While ".doc" files may contain any kind of character, RTF files only use printable characters, and clearly differentiate content from application commands. Thus RTF files can be laboriously read by humans without the aid of specialized applications. Nonetheless RTF is clearly a proprietary format: each new version of Word modifies and expands the grammar of RTF, and forces other applications to update their RTF filters to keep up with the innovations. Each expansion adds new terms to the RTF vocabulary, making it quite difficult to understand the exact meaning of every command. In fig. 3 I show the RTF version of the document shown previously.

```
{\rtf1\mac\ansicpg10000\uc1
\deff0\deflang1040\deflangfe1040{\upr{\fonttbl{\f0\fnil\fcharset256\fprq2{\*\
panose 02020603050405020304}Times New
Roman;}{\f4\fnil\fcharset256\fprq2{\*\panose 02000500000000000000}Times;}
}{\*\ud{\fonttbl{\f0\fnil\fcharset256\fprq2{\*\panose
02020603050405020304}Times New Roman;}{\f4\fnil\fcharset256\fprq2{\*\panose
02000500000000000000}Times;}}}}{\colortbl;\red0\green0\blue0;\red0\green0\blu
e255;\red0\green255\blue255;
\red0\green255\blue0;\red255\green0\blue255;\red255\green0\blue0;\red255\gree
n255\blue0;\red255\green255\blue255;\red0\green0\blue128;\red0\green128\blue1
28;\red0\green128\blue0;\red128\green0\blue128;\red128\green0\blue0;\red128\g
reen128\blue0;
\red128\green128\blue128;\red192\green192\blue192;}{\stylesheet{\widctlpar\ad
justright \lang1033\loch\af4\hich\af4\dbch\f4\cgrid \snext0 Normal;}{\*\cs10
\additive Default Paragraph Font;}}{\info{\title  }{\author Fabio}{\operator
Fabio}
{\creatim\yr2000\mo4\dy9\hr17\min36}{\revtim\yr2000\mo4\dy9\hr17\min36}{\vers
ion2}{\edmins0}{\nofpages3}{\nofwords72}{\nofchars412}{\nofcharsws505}{\vern1
17}}\paperw11906\paperh16838\margl1134\margr1134\margt1417\margb1134
\facingp\deftab708\widowctrl\ftnbj\aenddoc\hyphhotz283\margmirror\formshade\v
iewkind1\viewscale100\pgbrdrhead\pgbrdrfoot \fet0\sectd
\linex0\headery709\footery709\colsx709\endnhere\sectdefaultcl
{\*\pnseclvl1\pnucrm\pnstart1\pnindent720\pnhang{\pntxta .}}
{\*\pnseclvl2\pnucltr\pnstart1\pnindent720\pnhang{\pntxta
.}}{\*\pnseclvl3\pndec\pnstart1\pnindent720\pnhang{\pntxta
.}}{\*\pnseclvl4\pnlcltr\pnstart1\pnindent720\pnhang{\pntxta
)}}{\*\pnseclvl5\pndec\pnstart1\pnindent720\pnhang{\pntxtb (}{\pntxta )}}
{\*\pnseclvl6\pnlcltr\pnstart1\pnindent720\pnhang{\pntxtb (}{\pntxta
)}}{\*\pnseclvl7\pnlcrm\pnstart1\pnindent720\pnhang{\pntxtb (}{\pntxta
)}}{\*\pnseclvl8\pnlcltr\pnstart1\pnindent720\pnhang{\pntxtb (}{\pntxta
)}}{\*\pnseclvl9
\pnlcrm\pnstart1\pnindent720\pnhang{\pntxtb (}{\pntxta )}}\pard\plain
\qc\widctlpar\brdrt\brdrtnthsg\brdrw45\brsp20
\brdrl\brdrtnthsg\brdrw45\brsp80 \brdrb\brdrtnthsg\brdrw45\brsp20
\brdrr\brdrtnthsg\brdrw45\brsp80 \adjustright
\lang1033\loch\af4\hich\af4\dbch\f4\cgrid {\page
\par \par }{\fs72 \par
\par \hich\af0\dbch\af4\loch\f0 Three men in a boat
\par }{\fs48 \hich\af0\dbch\af4\loch\f0 To say nothing of the dog!
\par \par \par \par \par \hich\af0\dbch\af4\loch\f0 Jerome K. Jerome
\par \par \par \par \par \par \par \par \par \par \par
\par \hich\af0\dbch\af4\loch\f0 FV Books \par }{\fs20 \par \par \par
\par }\pard \qc\sl-20\slmult0\widctlpar\adjustright {\caps\fs34 \page
\par }\pard \qc\widctlpar\adjustright {\caps\fs34 \hich\af0\dbch\af4\loch\f0
Chapter 1
\par }\pard \li567\ri851\sb400\sa240\widctlpar\adjustright {\i\fs36
\hich\af0\dbch\af4\loch\f0 \hich\f0 Three invalids - Sufferings of George and
Harris - A victim to one hundred and seven fatal maladies -
\u8230\'c9\loch\f0
\par }\pard \qj\widctlpar\adjustright {\fs34 \hich\af0\dbch\af4\loch\f0
T}{\scaps\fs34 \hich\af0\dbch\af4\loch\f0 here}{\fs34
\hich\af0\dbch\af4\loch\f0  were four of us - George, and
Willia\hich\af0\dbch\af4\loch\f0
m Samuel Harris, and myself, and Montmorency. We were sitting in my room,
smoking, and talking about how bad we were - bad from a medical point of view
I mean, of course.
\par \hich\af0\dbch\af4\loch\f0 \hich\f0 We were all feeling seedy, and we
were getting quite nervous about it\u8230\'c9
\par }\pard \widctlpar\adjustright {\fs34 \par }}
```

Fig. 3: the RTF format of the same document

Another format that MS Word can save documents in is HTML: this is the standard language for hypertext documents available on the World Wide Web. HTML, derived from SGML and close relative to XML, is not proprietary, but an international standard: a proper body (the

World Wide Web Committee, or W3C) manages the development and evolution of the language, and there exist hundreds of applications that can manage this format. Thus HTML avoids the drawbacks of binary formats by being text-based (and thus it is easily readable), and the drawbacks of proprietary formats by being an international standard (and thus it has a controlled evolution and a large base of supporting software).

```
<HTML>
  <HEAD>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
    <META NAME="Generator" CONTENT="Microsoft Word 98">
    <TITLE>Three men in a boat</TITLE>
  </HEAD>
  <BODY>
      <P ALIGN="CENTER">&nbsp;</P>
      <FONT SIZE="7">
         <P ALIGN="CENTER">Three men in a boat</P>
      </FONT>
      <FONT SIZE="6">
         <P ALIGN="CENTER">To say nothing of the dog!</P>
         <P ALIGN="CENTER">&nbsp;</P>
         <P ALIGN="CENTER">Jerome K. Jerome</P>
         <P ALIGN="CENTER">&nbsp;</P>
         <P ALIGN="CENTER">FV Books</P>
      </FONT>
      <P ALIGN="CENTER">&nbsp;</P>
      <FONT SIZE="5">
        <P ALIGN="CENTER">Chapter 1</P>
      </FONT>
      <DIR><DIR>
         <FONT SIZE="5"><I>
               <P>Three invalids - Sufferings of George and Harris -
               A victim to one hundred and seven fatal maladies - ... </P>
         </I></FONT>
      </DIR></DIR>
      <FONT SIZE="5">
          <P ALIGN="JUSTIFY">There were four of us - George, and William
Samuel
           Harris, and myself, and Montmorency. We were sitting in my room,
           smoking, and talking about how bad we were - bad from a medical
           point of view I mean, of course. </P>
          <P ALIGN="JUSTIFY">We were all feeling seedy, and we were getting
           quite nervous about it...</P>
      </FONT>
  </BODY>
</HTML>
```

Fig. 4: A semiautomatic HTML version of the Word document.

Nonetheless, HTML is an application-oriented format: the commands that are present in a HTML file provide specific formatting instructions that are useful for on-screen visualization, but provide no support for more sophisticated uses. For instance, the HTML document shown in fig. 4 contains information about the requested alignment of the paragraphs and the font size of the text, but nothing that informs us that the string "Jerome K. Jerome" refers the novel's author, that the content is divided in chapters, or that the sentence "Three invalids - Sufferings of George and Harris - A victim to one hundred and seven fatal maladies - ..." is composed of several items of a summary of topics contained in the chapter. Applications different from on-screen browsers will find it hard to draw useful information from this document: for instance, indexing applications will not be able to determine the author or the content or the elements of the chapter summaries.

In this brief roundup, we have shown that text-based formats have advantages over binary formats in clearness and understandability, and that standard formats have advantages over proprietary formats in availability and stability. We have further described the disadvantages of application-oriented formats, that inhibit the creation of applications providing additional sophisticated functionalities over existing data. This has an evident impact on the preservability of such documents: most HTML documents, for instance, are optimized for displaying on color screens of a given resolution that are widely available currently. In a few years, we may require that these documents are displayed on the smaller screens of cellular phones, or are read by a synthesized voice in a car-radio system for drivers. Both these applications will make little use of instructions detailing alignment or font size, and would rather require a different class of specifications. Fixing the content of our documents with HTML tags, therefore, will seriously impede the preservation of our documents in just a few years. In the next section we will examine how these problems can be avoided with a text-based, standard, data-oriented format: XML.

What is XML for?

In some sense, XML (Extensible Markup Language - *sic*!) can be seen as the ultimate answer to our quest for preservable data formats. XML [7] provides a solution to all the issues raised in the previous section, and has advantages way beyond those connected with preservation. Its generality, its sophistication, and the widespread industry support, all contribute to make XML one of the most interesting products of computer science.
The most interesting aspect of XML for the purposes of our discussion is that XML is not a data format in the sense that I have been meaning so far: that is, it is not a specific dictionary of terms (or words, commands, bytes) wrapping the content and providing processable information about the content itself. Rather, XML is a meta-format, a common syntax to create and express any dictionary of terms about the content of our documents.
XML looks like HTML, and can be easily mistaken for it. Although there are a few syntactical differences, they both use tags, attributes, and entities variously structured and nested to label the important elements of a document. But while HTML is a predefined and closed set of tags and entities, which we must force our documents to fit to, XML is a way to create any set of tags and entities, so that we can create and customize them for the particular characteristics of our documents. In simpler words, XML is an HTML where I am allowed to decide the tag names. This apparently simple difference has a great impact on our ideas about marking up documents.
A simple way to describe XML would be a fast and simple overview of its syntactical characteristics: XML provides an HTML-like syntax for document elements where the actual names are not predefined, but can be decided at will. That is to say, XML maintains the peculiar look of tags, attributes and entities made famous by HTML, but it also allows us to use any name for these tags, atributes and entities. Because of this, even HTML names can be considered valid XML names, so that the following can be considered a "correct" chunk of information in both XML and HTML:

```
<P align="right">This is a correct document element</P>
```

Syntactically, XML is more rigorous than HTML, requiring strict nesting of tags, fully quoted attribute values and proper ending of document elements (ending tags are not optional in XML as they are in many cases in HTML). Nonetheless; many HTML documents are already or can be easily transformed into correct XML documents. Indeed, the HTML document in fig. 4 is already a correct XML document (but had to be corrected by hand from the one automatically created automatically by MS Word).
But it is in its ideology, that XML mostly differs from HTML. XML brings forth the idea of generic markup, a radically different approach to document markup. The possibility of deciding the names for tags, attributes and entities means that the author has an additional degree of freedom in the creation of documents, and that he/she can tailor the markup of our documents to any needs. Indeed, he/she can tailor it to *all* needs. Generic markup, in fact, insists that markup should not be added with a specific application in mind (say, printing), but

that it should describe generically the document, i.e., it should describe the document for what it is, rather than for what it is used for.
For instance, it is a mere accident, for a block of text within a document; to be formatted with a given font or a given margin width, or even to be a paragraph. But sections, chapters, titles, captions, etc. will always remain such, regardless of the style transformation they will be subjected to. Fig. 5 shows a possible generic markup for our document using the XML syntax.

```
<?xml version="1.0"?>
<!DOCTYPE NOVEL SYSTEM "novel.dtd">

<NOVEL>
  <FRONT_MATTER>
    <TITLE>Three men in a boat</TITLE>
    <SUBTITLE>To say nothing of the dog!</SUBTITLE>
    <AUTHOR>Jerome K. Jerome</AUTHOR>
    <PUBLISHER>FV Books</PUBLISHER>
  </FRONT_MATTER>
  <CONTENT>
    <CHAPTER>
      <TITLE>Chapter 1</TITLE>
      <SUMMARY>
         <ITEM>Three invalids</ITEM>
         <ITEM>Sufferings of George and Harris</ITEM>
         <ITEM>A victim to one hundred and seven fatal maladies</ITEM>
         ...
      </SUMMARY>
      <BODY>
          <PARA>There were four of us - George, and William Samuel
            Harris, and myself, and Montmorency. We were sitting in my room,
            smoking, and talking about how bad we were - bad from a medical
            point of view I mean, of course. </PARA>
          <PARA>We were all feeling seedy, and we were getting quite nervous
          about it ... </PARA>
      </BODY>
    </CHAPTER>
    <CHAPTER>
      ...
    </CHAPTER>
  </CONTENT>
</NOVEL>
```

Fig. 5: One of the possible XML versions of the document

While maintaining several similarities with HTML (particularly the use of tags and attributes), it is clear that the XML version of the document pays no attention to layout and typographical issues, but rather that it tends to describe the intrinsic structure of the document: where HTML has lots of FONT tags and ALIGN attributes, which are only useful for graphical rendering, the XML version uses tags like CHAPTER, TITLE, etc., i.e., it provides a structural classification of the parts of the document.
XML exhibits all the good characteristics of long-lasting data formats, as discussed in the previous section:

· XML is standard: it is not proposed by a commercial enterprise, but by an organization of standards, the World Wide Web Committee, and it has been widely accepted by hundreds of different organizations.
· XML is text-based: it allows the visual exploration of the internals of a stored document even in the absence of the specific application that created it.
· XML is data-oriented, rather than application-oriented. The ideology behind XML asks the markup to describe facts about the document, rather than the application that was meant to manage it.

These aspects cooperate to provide realistic expectations of preservability of XML documents: it is reasonable to expect at least some of these hundreds of organizations to keep on supporting the format for a long time; it is reasonable to expect, in a far future where specialized applications have vanished, to be able at least to read the documents with a generic tool, and be able to tell the markup from the content. It is reasonable to expect that we will still be able to understand the purpose and meaning of the markup, and therefore to create tools that will let us re-use the content in a meaningful way.

Virtual documents

The World Wide Web has taken off as a mass medium with a speed unprecedented in the history. Already most of the organization with content or image to present to the big public has opened a site and made available its data. Both as information system for its internal purposes (the Intranet), and as advertisement or customer support system for the users at large, every commercial organization of some size (and many of no size at all), have created on line shops, manuals, and portals. The Internet is the place where image, support and sales converge to create a unique customer experience.

Already the Web holds more data than the Library of Congress. Soon it will hold more stuff than the sum of all the collections of data in the world [3]. Soon, for some information to be on the Web will just be the normal, obvious choice, and the decision to put it elsewhere besides the Web will be just an option with explicitly considered advantages and drawbacks.

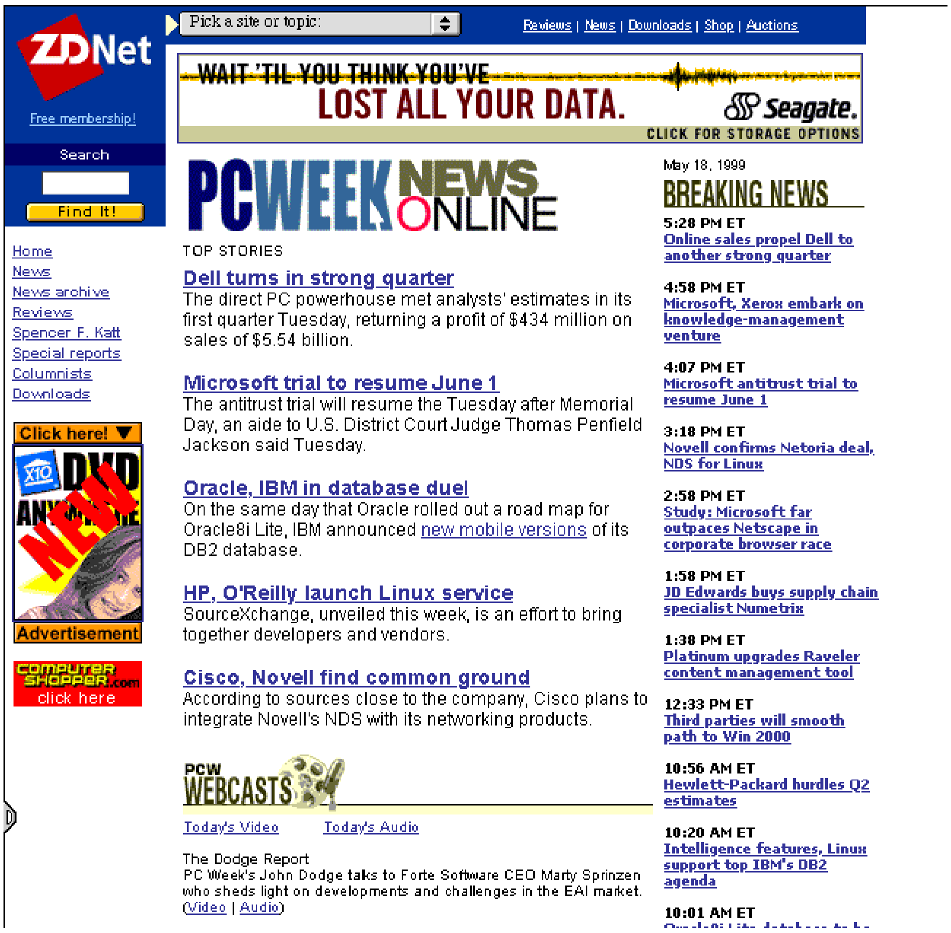
Even if most of the people on the Internet are considering it little more than an amusement, and most of the commercial organizations are seeing it little more than a advertising opportunity, there are huge quantities of data that are worth preserving out there. Yet the very rapid development of the Internet will create several problems when the issue will arise of preservation of this huge amount of data. One of the most common experience of the early Web users was the infamous "404 Not found" error, labeling a link as a dead end, the resource being pointed to disappeared and never to be available again. Several initiatives have started looking for a solution to this.

In the easiest case, the information has just moved. A restyling of the site, an update of the information available, and its URL is no longer valid. That is to say, the document is still out there somewhere, but the address I wrote down has changed. In fact, the URL contains several locally decided information (for instance path and filename) that may change without warning. The URN initiative [8] plans to provide stable and reliable network addresses to resources. The only way to do so is to plan a resource name that has no bearing with its physical accessing mechanism (domain name, directory, filename, etc.), and to provide a way to map the name to the current URL. We would then store in our bookmarks and in our links the stable URN, and whenever we need to activate it, the browser would request the current specific URL and access the resource.

Sometimes, on the other hand, the resource just disappears. In 1994 Netscape Communicator decided to introduce a scripting language for HTML pages, and in the beginning of 1995 a big quantity of documents, manuals and white papers appeared on its site to lure programmers and web designers into using the Livescript language. More or less in the same period, the sudden and huge success of Java (which started as a completely independent initiative from Sun Microsystem) led the marketing managers of Netscape to decide to rename the scripting language. All documents about Livescript were changed overnight, and Javascript became the scripting language of Netscape. No technical change was done, just a renaming. Yet, no mention of the name "Livescript" was left on the site, with a complete rewriting of the past, at least on the site of Netscape. Many similar events have happened on the Web. When an organization is in complete control of the diffusion of the data it makes available, it is very easy to rewrite it to suit the strategies of the moment.

The idea of archiving the data in spite of the owner's approval is tempting. Brewster Kahle's Internet Archive (`http://www.archive.org`) is based on this idea [9]. An automatic process (a robot) systematically requests every page available on the Web; and systematically stores it

on a local disk. Every time a stored page changes, the robot will ask for the new version, and will store it side by side with the older ones. The purpose of this section of my paper is to explain why, in my view, this task, tantalizing though it may seem, is intrinsically doomed.
Of course, given the amount of information that is routinely put on the web, this task can easily seem gigantic. But it is my conviction that, even if provided with enough resources (disk, computing power, etc.), this task is intrinsically impossible. The main reason for this are *virtual documents* [10].
I have already mentioned that text-based data formats are extremely interesting for the variety of applications that they allow. Virtual documents were previously unheard of, but thanks to HTML they have become extremely common. A virtual document for our purposes could be defined as a document whose persistent state is different from its perceived state. For instance, a view on a database (say, the output of a query) is a document that doesn't exist as such, but is built on demand as the output of the query. The content of the document, in this case, is less important than the query that generated it. Another example is a compound document that is built from different chunks of data immediately before being displayed to the user, or a document that is built by an application as its output. The most evident cases are portals, dynamic collections of information chunks, links and services. In fig. 6 I show an example of a virtual document.



TOP STORIES

Dell turns in strong quarter
The direct PC powerhouse met analysts' estimates in its first quarter Tuesday, returning a profit of $434 million on sales of $5.54 billion.

Microsoft trial to resume June 1
The antitrust trial will resume the Tuesday after Memorial Day, an aide to U.S. District Court Judge Thomas Penfield Jackson said Tuesday.

Oracle, IBM in database duel
On the same day that Oracle rolled out a road map for Oracle8i Lite, IBM announced new mobile versions of its DB2 database.

HP, O'Reilly launch Linux service
SourceXchange, unveiled this week, is an effort to bring together developers and vendors.

Cisco, Novell find common ground
According to sources close to the company, Cisco plans to integrate Novell's NDS with its networking products.

The Dodge Report
PC Week's John Dodge talks to Forte Software CEO Marty Sprinzen who sheds light on developments and challenges in the EAI market. (Video | Audio)

May 18, 1999
BREAKING NEWS

5:28 PM ET
Online sales propel Dell to another strong quarter

4:58 PM ET
Microsoft, Xerox embark on knowledge-management venture

4:07 PM ET
Microsoft antitrust trial to resume June 1

3:18 PM ET
Novell confirms Netoria deal, NDS for Linux

2:58 PM ET
Study: Microsoft far outpaces Netscape in corporate browser race

1:58 PM ET
JD Edwards buys supply chain specialist Numetrix

1:38 PM ET
Platinum upgrades Raveler content management tool

12:33 PM ET
Third parties will smooth path to Win 2000

10:56 AM ET
Hewlett-Packard hurdles Q2 estimates

10:20 AM ET
Intelligence features, Linux support top IBM's DB2 agenda

10:01 AM ET

Fig. 6: PCWeek online: an example of a virtual document

In all these cases, the important fact is that the user sees "less than" the real document. He only gets the result, not the machinery to compile the result. The document received by the browser in fig. 6, for instance, is not stored in this form anywhere on the origin server. In fact, as it often happens, the document is built on demand as soon as I have requested it.

Thanks to HTML and its text-based format, it is very easy for an application to format its output so that it is immediately accessible on the Web. Usually an empty HTML template contains just the main structure of the page, with predefined slots for the actual content. When requesting the document, a server-side engine collects from a database the content, and applies it to the template to generate on the fly the final document sent to the user.
Preserving the final result is pointless. The analogy to magazines and newspaper does not hold: even though the newspapers are compound documents that are the result of intense copy and paste activities by those who create it, a newspaper still has fixed content and a regular issuing periods which can be relied on for archival. A web virtual document, on the other hand, can be updated several times per hour, and indeed continuously, and in many cases is tailored to the preferences and interests of the person requesting the document. Furthermore, random processes can be added to let each access differ from the previous ones, thereby enhancing the sense of novelty and the site appeal. Therefore, there is no sense in storing the document as received on the browser, well knowing that in a few minutes the same document could very well have a considerably different content, depending on the user requesting it, the amount of new information added in the meantime, or even the random result of a dice throw.
In order to archive a traditional database, it would make no sense to express all the possible queries and store the result. The database is usually considered as a whole, and archived as such. Analogously, it makes no sense whatsoever to archive the pages that are generated by a process such as the one just described. It is necessary to have access to the actual text database, to the HTML templates, and even to the actual engine that creates the perceived documents. Only by the correct archival of these three actors we can safely assume we are preserving one such site for the future.
And, of course, archiving the database and the rest of the data cannot be done unless the owners of the data itself cooperate. The intent to archive the Internet in the absence of collaboration by the actual owners of the data, therefore, is clearly futile.

Conclusions

This paper does not end with a hopeful note: the issue of preserving digital documents is a complex one, and not easily solved. Although we may have learnt from our past mistakes, the technology progresses, and creates new challenges that cannot be tackled with old approaches.
Although XML and its cohort of languages can provide important advancements in the correct labeling and structuring of information, the current trend in Web technologies seems to favor complex, dynamic collections of information that are impossible to define, and even less to archive.
It is hard to say how we should manage, and , frankly, even *whether* we should. Preserving data meant to be consumed fresh may well be considered an unacceptable imposition over the will of the authors of the data. Currently, the cooperation of the owner of the sites is indispensable in the preservation of complex data, especially virtual, dynamic documents. It is possible that this is just fair, that the owners should decide whether and what should be preserved and what should be destined to oblivion.

Acknowledgement

I wish to thank Paolo Ciancarini, Giulio Blasi, Dino Buzzetti, Andrea Angiolini and Andrea Babich for the interesting discussions about the topic of electronic preservation, that have lead me to write this paper. Giulio Blasi deserves additional thanks for the impressive conference he organized on this topic, "The future of memory" (San Marino, May 1999), and for the patience he has shown in waiting for my contribution. And many, many thanks go to Isabella, who has kept me to writing this paper.

Bibliography

[1] Stewart Brand, *The clock of the long now, Time and responsibility*, Basic Books, 1999

[2] Task Force on Archiving of Digital Information, *Preserving Digital Information*, a report commissioned by the Commission on Preservation and Access and the Research Libraries Group, 1996, `http://www.rlg.org/ArchTF/tfadi.index.htm`

[3] Michael Lesk, *How much information is there in the world?*, Time & Bits Conference, 1999, `http://www.longnow.org/10klibrary/TimeBitsDisc/ksg.html`

[4] Andrea Babich, *Retrogaming, un caso di digital preservation dei videogiochi*, Master Thesis, Communication Sciences, University of Bologna, 2000 (in italian)

[5] Multiple Arcade Machine Emulator (MAME), `http://www.mame.net`

[6] Steve De Rose, *Where did my bytes go?*, 1999 `http://www.stg.brown.edu/~sjd/mymusings/datasurvival.html`

[7] T. Bray, J. Paoli, C. M. Sperberg-McQueen, *Extensible Markup Language, (XML) 1.0*, W3C Recommendation 10 February 1998, `http://www.w3.org/TR/REC-xml`

[8] K. Sollins, L. Masinter, Functional Requirements for Uniform Resource Names, RFC 1737, Internet Engineering Task Force (IETF), December 1994, `http://www.ietf.org/rfc/rfc1737.txt`

[9] Brewster Kahle, *Preserving the Internet*, Scientific American, March 1997, `http://www.sciam.com/0397issue/0397kahle.html`

[10] M. Milosavljevic, F. Vitali, C. Watters, Proceedings of the Workshop on Virtual Documents, Hypertext Functionalities and the Web, VIII Int. World Wide Web Conference, Toronto, Canada, 1999, `http://www.cs.unibo.it/~fabio/VD99/`