Università degli studi di Bologna

FACOLTA' DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Informatica

UN SISTEMA DI CREAZIONE DINAMICA DI DOCUMENTI INTERATTIVI IN PDF

Tesi di Laurea di:
Andrea Gentilini

Relatore:
Chiar.mo prof.
Paolo Ciancarini

3ª Sessione

Anno Accademico 2001-2002

INDICE

1. SOMMARIO

Scopo di questa tesi è studiare come sia possibile creare documenti attivi e interattivi in formato PDF (Portable Document Format), attraverso alcuni esempi di programmi.
Il lavoro consiste nella progettazione di alcuni prototipi che mostrano su schermo un problema del gioco degli scacchi. Tramite l'utilizzo della libreria "pdflib.h", creata da Thomas Merz nel 1997, è stato possibile avere come output dei file eseguibili e dei file PDF.
Il lavoro è stato svolto in ambiente Windows, tramite l'utilizzo del linguaggio di programmazione C.
Il primo esempio che viene presentato, mostra la creazione di un file PDF interattivo. Il documento è composto da una serie di pagine presentate in modalità "fullscreen". L'interattività viene resa tramite l'utilizzo di collegamenti ipertestuali fra le varie pagine del documento, il capitolo N spiega passo a passo le parti di codice sorgente importanti per la realizzazione dello scopo, spiegando mano a mano le funzioni della libreria "pdflib.h" usate.
Successivamente viene modificato il codice sorgente del primo file, per costruire un documento PDF attivo. Per documento PDF attivo si intende un documento che cambi pagina automaticamente dopo che è passato un po' di tempo (alcuni secondi); quindi in sostanza ho cercato di creare un documento PDF che presenti il problema scacchistico preso in considerazione, attraverso uno slide-show.
Il terzo documento creato, riprende il codice sorgente dei due presentati precedentemente, ed è a sua volta interattivo. Questa volta non viene preso in considerazione solo il file PDF creato, ma viene messo in evidenza anche l'eseguibile ad esso associato, in quanto all'interno del documento creato è presente un link, appunto, al file eseguibile. Il file eseguibile è creato in modo tale che un utente, che apre il documento creato e clicca sul link che

rimanda all'eseguibile, crea un nuovo file in formato PDF che contiene le pagine successive del documento creato.
In conclusione si cercherà di analizzare i tre documenti creati sotto due punti di vista: la portabilità di tali documenti nei vari sistemi operativi e la grandezza degli stessi, cioè lo spazio su disco occupato.

2. INTRODUCTION TO PDF

2.1. History

In 1985, with the introduction of the first Apple LaserWriter, Adobe introduced PostScript, a language with powerful graphical abilities in order to specify the content of one page to print.
A page is viewed like a combination of several elements:

- basic geometrical objects, such as lines, rectangles, circles and curves, design by "pens" with tips of variable dimensions
- objects character in several typographical shapes (font), of whichever color, dimension and guideline
- bitmap, that is described figures do not given by an objects combinations but from rectangular combinations of pixel

N. B.
PostScript documents are not made in order to be editable: they are sequences of instructions for the press.

The introduction of personal computers into the business has drastically changed the archiving environment. Prior to the 1990's, most offices still had typing pools and world processing groups and kept records on paper in centralized files. But once computers become the norm for the majority of workers, the usefulness of the centralized file room disappeared. It become everyone's responsibility to create, file, and maintain his or her own documents. As a result, corporations and governments have lost control over these records.
Establishing and maintaining an electronic archive require policy decisions, procedures, and organization – wide planning along with a commitment to follow the organizational standards.
The requirements for the adequacy of records are determined by each organization's internal business and legal needs, as well as

external regulations or requirements. Thus, the requirements for each organization will be different.
Subsequently, in 1992, John Warnock, co-founder of Adobe Systems Incorporated, speaking about the goals of a development project known as Camelot, said, “There is no universal way to communicate and view this printed information electronically... What industries badly need is a universal way to communicate documents across a wide variety of machine configurations, operating systems, and communication networks”.
The Camelot project developed the technology known as PDF.
In 1993, the first PDF specification was published at the same time the first Adobe Acrobat products were introduced.
At that time, the PostScript page description language was rapidly becoming the worldwide standard for the production of the printed page; PDF builds on the PostScript page description language by layering a document structure and interactive navigation features on PostScript’s underlying imaging model, providing a convenient, efficient mechanism enabling documents to be reliably viewed and printed anywhere.
In other terms, PDF leveraged the ability of the PostScript language to render complex text and graphics and brought this feature to the screen as well as the printer. Over the post 8 years, aided by the explosive growth of the Internet, PDF has become the “de facto” standard for the electronic exchange of documents. Well over 200 million copies of the free Acrobat Reader application have been distributed around the world, facilitating efficient sharing of digital content.

2.2. PDF and the PostScript language

PDF is based in great part on PostScript. But while the latter one is planned in order to describe sequences of physical pages, that is

documents to print, PDF is planned in order to describe set of pages, that are documents to deliver on the net. Moreover, a PDF file can contain not only information of the graphical aspect of one page, but also of its "behaviours" and the information contained in the document (metadata).
A PostScript file is a “program”, a true application that is created, transmitted and executed by an interpreter RIP (Raster Image Processor) PostScript.
The result of the execution of a PostScript document is the rastering of one bitmap normally on paper or film. The objects that describe the graphical elements are difficult to analyze or to extract from the document.
To edit PostScript code is possible but much difficult: in kind it is better to return to the application that has created such code. A PDF file is in short a PostScript file already interpreted from a RIP and transformed in graphical objects good defined.
These objects are visible on screen, but their code is binary and incompressible. Since the moment that such objects are the result of a rastering operation, are more reliable for print respect to .ps or .eps files. For more this function it allows to observe on screen the obtainable result without sending the document to the printer.
Unlike PostScript, PDF is not a full – scale programming language; it trades reduced flexibility for improved efficiently and predictability. PDF therefore differs from PostScript in the following significant ways:

- PDF enforces a strictly defined file structure that allows an application to access parts of a document in arbitrary order.
- To simplify the processing of content streams, PDF does not include common programming language features such as procedures, variables, and construct.
- PDF file contain information such as font metrics to ensure viewing fidelity.

- A PDF file may contain additional information that is not directly connected with the imaging model, such as hypertext links for interactive viewing and logical structure information for document interchange.

2.3. PDF format

The term Portable Document format, or PDF, was coined to illustrate that a file conforming to this specification can be viewed and printed on any platform – UNIX, Mac Os, Microsoft Windows, and several mobile devices as well – with the same fidelity. A PDF document is the same for any of these platforms. It consists of a sequence of pages, with each page including the text, font specifications, margins, layout, graphical elements, and background and text colors. With all this information present, the PDF file can be imaged accurately for the screen and the printing device. It can also include other items such as metadata, hyperlinks, and form fields.
A PDF document is a set of objects that together describe the appearance of one or more pages.

2.4. Properties

This section describes some important properties of PDF.

2.4.1. Imaging model

At the heart of PDF is its ability to describe the appearance of sophisticated graphics and typography. This is achieved through the use of the Adobe imaging model, the same high-level, device-

independent representation used in the PostScript page description language.
A high-level imaging model enables application to describe the appearance of pages containing text, graphical shapes, and sample images in terms of abstract graphical elements rather than directly in terms of device pixel. Such a description is economical and device-independent, and can be used to produce high-quality output on a broad range of printers, displays, and other output devices.

2.4.2. Portability

PDF files are represented as a sequences of 8-bit binary bytes. A PDF file is designed to be portable across all platforms and operating systems. The binary representation is intended to be generated, transported, and consumed directly, without translation between native character sets, end-of-line representations, or other conventions used on various platforms.

2.4.3. Font management

Managing fonts is a fundamental challenge in document interchange. Generally, the receiver of a document must have the same fonts that were originally used to create it. If a different font is substituted, its character set, glyph shapes, and metrics may different from those in the original font. This can produce unexpected and undesirable results, such as lines of text extending into margins or overlapping with graphics.
Font management, in PDF, is primarily concerned with producing the correct appearance of text-that is, the shape and placement of glyphs.
PDF provides various means for dealing with font management: it supports various font formats, including Type 1, TrueTipe, and

CID-keyed fonts; and contains a font descriptor for each font that it uses. The font descriptor includes font metrics and style information, enabling a viewer application to select or synthesize a suitable substitute font if necessary.

2.4.4. Security

PDF has two security features that can be used, separately or together, in any document; the document an be encrypted or can be digitally signed to certify its authenticity.
If the document is encrypted, only authorized users can access it. There is separate authorization for the owner of the document and for all other users; they can be selectively restricted to allow only certain operations, such as viewing, printing, or editing.

2.5. Creating PDF files

There are several ways of creating PDFs. They differ in the software required, functionality, and simplicity. The following section is a small introduction to the most important variants.

2.5.1. Acrobat PDF Writer

PDF Writer offers the simplest way of creating PDFs. It is installed at system level as a printer driver under Windows or on the Mac and makes all programs with a print function PDF capable.
PDF Writer is not based on PostScript, but on the graphics interface of the respective operating system – QuickDraw on the Mac or GDI under Windows. These imaging models are not powerful as PostScript.

PDF Writer has a whole load of functional weakness. The most serious of these is that it can not process ESP (Encapsulated PostScript) files satisfactorily.
Since the PDF driver does not interpret PostScript files, it only embeds the screen preview – a bitmap version of the graphic – in the PDF file which produces. Because this EPS preview is rasterized at low resolution, it will not usually be of acceptable quality.

2.5.2. Acrobat Distiller

Distiller is the method of choice for conversion to PDF where requirements are higher. This program contains a full-blown PostScript LEVEL 2 interpreter. This means that Distiller accepts the same page description file as a printer or an imagesetter. Unlike these output devices, Distiller does not rasterize the data (i.e. does not generate a pixel image) but converts the PostScript page description to a PDF file and stores it on the hard disk.
While PDF Writer makes a large number of programs "PDF capable", Distiller brings all programs which can create PostScript output into the PDF sphere. This also makes it possible to create PDF files on those platforms (Unix for example) where there are no system level printer drivers. Apart from "normal" PostScript instructions, Distiller also makes use of the pdfmark operator. This extension makes it possible to define PDF features in the PostScript code which would otherwise have to be added later using Exchange. These features include link, bookmarks, article threads, and many more. In addition, Distiller offers several options for processing output files at the prepress stage, for example to control imagesetters or digital printing presses.

2.5.3. Other methods to create PDF files

All of the methods for generating PDF files described previously rely on Acrobat. However, there are also ways of generating PDF files without Acrobat software. More and more graphics and image processing applications include export modules for PDF. These export modules often do not support full PDF functionality so there are limitations on the files produced.

There are many libraries for programming PDF files, such as PDFlib and Itext. They support many languages, such as C, Perl and Java.

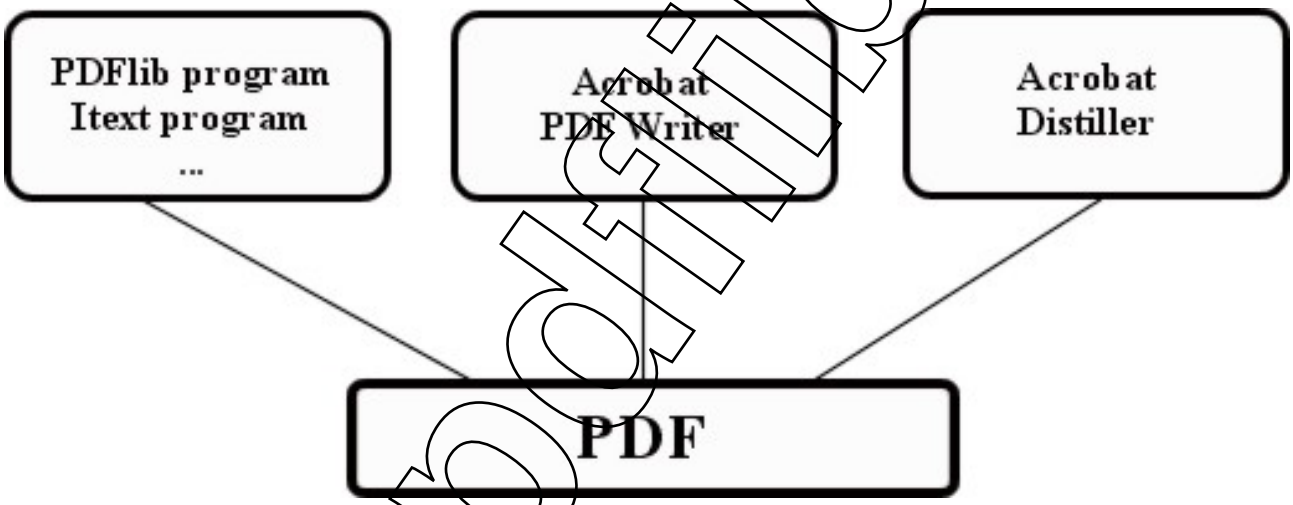

fig. 1: Various paths to PDF

3. INTRODUCTION TO PDFlib.h

3.1. What is PDFlib

PDFlib is a library written by Tomas Merz in 1997, which allows one to generate files in Adobe's Portable Document Format (PDF). PDFlib acts as a backend to your own programs. While you (the programmer) are responsible for retrieving or maintaining the data to be processed. PDFlib takes over the task of generating the PDF code which graphically represents your data. While you must still format and arrange your text and graphical objects, PDFlib frees you from the internals and intricacies of PDF. PDFlib offers many useful functions for creating text, graphics, images, and hypertext elements in PDF.

3.2. PDFlib support

PDFlib is available on a variety of platforms, including Unix, Windows, Mac Os, and EBCDIC-based systems such as IBM eServer iSeries 400 and zSeries S/390. PDFlib itself is written in the C language, but it can be also accessed from several other languages and programming environments which are called language bindings. These language bindings cover all major Web application languages currently in use. The Application Programming Interface (API) is easy to learn, and is identical for all bindings. Currently the following bindings are supported:

- ActiveX/COM, providing access from Visual Basic, Active Server Pages with VBScript or Jscript, Allaire ColdFusion, Borland Delphi, windows Script Host, and many other environments

- ANSI C
- ANSI C++
- Java, including servlets
- PHP hypertext processor
- Perl
- Python
- RPG (IBM eServer iSeries 400)
- Tcl

3.3. PDFlib language bindings

While C programming language has been one of the cornerstones of system and application software development for decades, a whole slew of other languages have been around for quite some time which are either related to new programming paradigms (such as C++), open the door to powerful platform - independent scripting capabilities (such as Perl, Tcl, and Pyton), promise a new quality in software portability (such as Java), or provide the glue among many different technologies while being platform – specific (such as ActiveX/COM).
Naturally, the question arises hot to support so many languages with a single library. Fortunately, all modern language environments are extensible in some way or another. This include support for extensions written in the C language in all cases. Looking closer, each environment has its own restrictions regarding the implementation of extensions.

3.4. PDFlib features

The PDFlib API offers the following major features:
- PDF documents of arbitrary length and page formats

- Text output in different fonts
- The ability to embed PostScript font descriptions
- Common vector graphics primitives – lines, curves, arcs, rectangles, etc.
- Read PostScript font metrics from AFM files
- Process common graphics file formats, e.g. TIFF, GIF, JPEG
- Generate hypertext elements such as bookmarks
- Features supported in PDF but not accessible in Acrobat software, e.g. page transition effects like shades and mosaic

All of these may be achieved by using a simple API without the application programmer being directly involved with PDF objects or operators.

3.5. PDFlib targets

PDFlib's primary target is creating dynamic PDF on the World Wide Web. Similar to HTML pages dynamically generated with a CGI script on the Web server, you may use a PDFlib program for dynamically generating PDF reflecting user input or some other dynamic data, e.g. data retrieved from the Web server's database.
Equally important are all kinds of converters from X to PDF, where X represents any text or graphics file format. Again, this replaces the sequence X - PostScript - PDF with simply X – PDF, which offers many advantages for the some common graphics file formats like GIF or JPEG. Using such a converter, batch converting lots of text or graphics files is much easier than using the Adobe Acrobat suite of programs. Several converters of this kind are supplied with the library.
In addition to generating PDF documents on a file, PDFlib can also be instructed to generate the PDF directly in memory (in – core). This technique offers performance benefits since no disk –

based I/O is involved, and the PDF document can, for example, directly be streamed via HTTP. You may, at your opinion, periodically collect partial data, or fetch the complete PDF document in one big chunk at the end. Interleaving production and consumption of the PDF data has several advantages. Firstly, since not all data must be kept in memory, the memory requirements are reduced. Secondly, such a scheme can boost performance since the first chuck of data can be transmitted over a slow link while the next chunk is still being generated. However, the total length of the generated data will only be known when the complete document is finished.

3.6. Programming with PDFlib

PDF's default coordinate system is used within PDFlib. The default coordinate system (or default user space in PDF lingo) has the origin in the left corner of the page, and uses the DTP point as unit:
1 pt = 1 inch / 72 = 25.4 mm / 72 = 0.3528 mm
The first coordinate increases to the right, the second coordinate increases upward. PDF client programs may change the default user space by rotating, scaling, translating, or skewing, resulting in new user coordinates. If the user space has been transformed, all coordinates in graphics and text functions must be supplied according to the new coordinate system. The coordinate system is reset to the default coordinate system at the start of each page.
In order to assist PDFlib users in working with PDF's coordinate system, the PDFlib distribution contains the PDF file "grid.pdf" which visualizes the coordinates for the several common page size (A4, A5, letter, etc..).
PDFlib functions are subdivided in 6 categories :

- General functions (PDF_open, PDF_close, etc.)
- Text functions (PDF_show, PDF_set_font, etc.)
- Graphics functions (PDF_translate, PDF_circle, etc.)
- Color functions (PDF_setcolor, etc)
- Image functions (PDF_open_image, PDF_close_image, etc.)
- Hypertext functions (PDF_add_locallink, PDF_add_pdflink, etc.)

Most PDFlib functions are subject to certain ordering and nesting constraints which are derived from their contribution to the generated document. Most of these constraints are rather obvious. For example, you must begin a page before you can close it. In the same spirit, the functions for opening a PDF document and closing it must always be paired. PDFlib uses a strict scoping system for defining and verifying the correct ordering of functions used by client programs. The function descriptions reference these scopes; the scope definitions can be found in the table below.

Table 1. Function scope definitions

scope name	***definition***
path	started by one of PDF_moveto( ), PDF_circle( ), PDF_arc( ), PDF_arcn( ), or PDF_rect( ) terminated by one of PDF_endpath(), PDF_fill_stroke()
Page	between PDF_begin_page( ) and PDF_end_page( ), but outside of path scope
Template	between PDF_begin_template( ) and PDF_end_template( ), but outside of path scope
pattern	between PDF_begin_pattern( ) and PDF_end_pattern( ), but outside of path scope
document	between PDF_open_*( ) and PDF_close( ), but outside of page, template, and pattern scope
object	in Java: the lifetime of the pdflib object, but outside of document scope; in other bindings between PDF_new( ) and PDF_delete( ), but outside of document scope
Null	outside of object scope
Any	when a function description mentions »any« scope it actually means any except null, since a PDFlib object doesn't even exist in null scope.

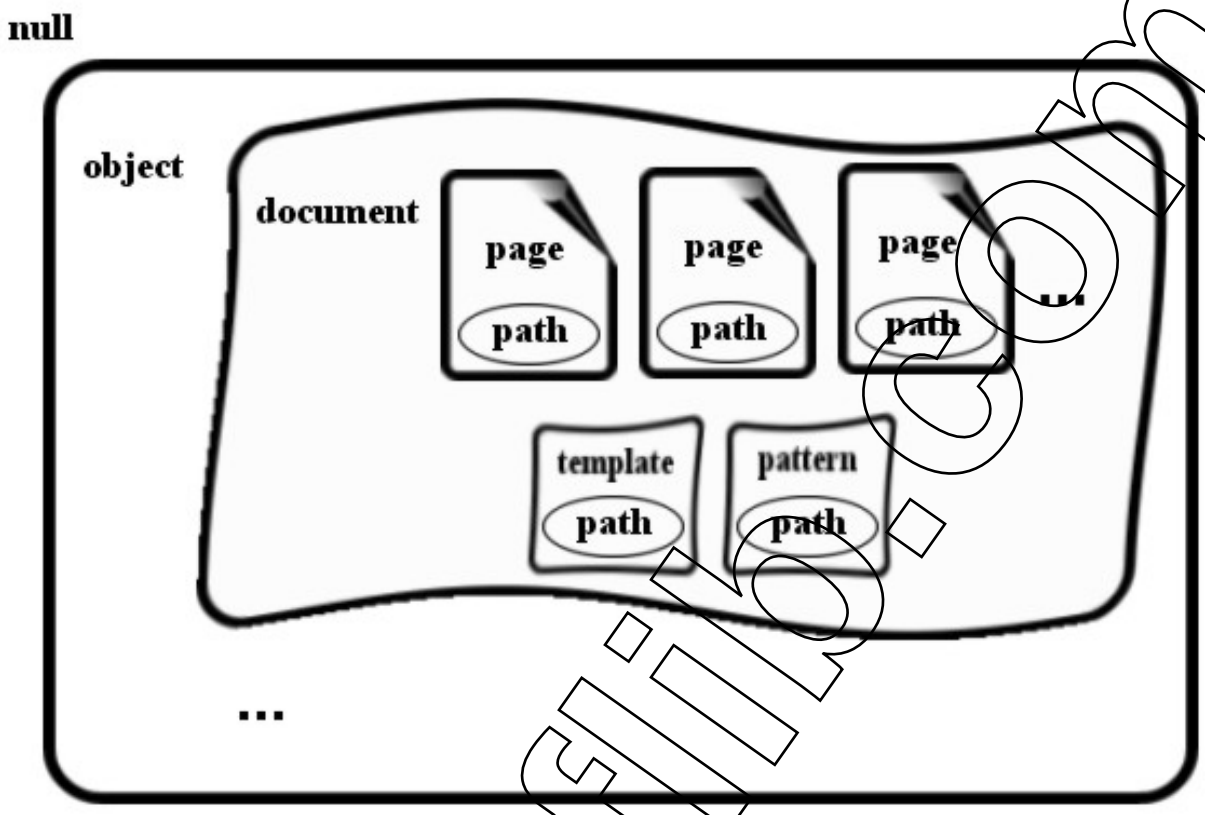

Fig 2: Relationship of scopes

4. INTRODUZIONE AI DOCUMENTI ATTIVI E INTERATTIVI

Dopo aver visto una breve introduzione al formato PDF e alla libreria Pdflib.h, occorre dare una definizione ai termini “attivo” e “interattivo”, essendo lo scopo di questo lavoro di tesi la costruzione di file appunto, attivi e interattivi.
In informatica esistono varie definizioni di file attivo, a seconda del contesto in cui si lavora. In generale esso può essere definito come un file che è stato prelevato dall'hard disk (o da altra unità di memoria di massa) e trasferito temporaneamente nella RAM per poterlo utilizzare; oppure come un file appena creato, ma non ancora salvato. Un file attivo, può anche essere definito come un documento che può fornire un certo comportamento autonomo: lo stesso documento può essere visualizzato, stampato, cercato, effettuando calcoli, producendo documentazione supplementare, effettuando animazioni…
In questo lavoro di tesi, un file PDF attivo viene definito come un documento che fornisce un comportamento autonomo, presentando una specie di “slide-show”.

Fig 3: Rappresentazione del rapporto fra file attivo e utente

L’interattività, in informatica, può essere definita come una modalità che consente uno scambio d’informazioni fra gli elementi differenti di un certo ambiente, in particolare tra un sistema d’elaborazione e un utente. Il termine “interattivo” si applica ad un programma che richiede la collaborazione dell'utente oltre che del computer. Ogni comando dell'utente comporta una risposta del calcolatore.
Adobe definisce una forma interattiva di un file PDF come un documento che ha una o più funzionalità supplementari incluse nel file. Queste funzionalità supplementari permettono che l’utente compia una delle operazioni sottostanti, mentre sta osservando un file PDF nel web browser o nel lettore Acrobat:

- Digitare i dati nei campi forniti dal documento
- Fare dei calcoli usando i campi forniti sul documento
- Navigare all’interno del documento, o passare ad altri documenti, o siti
- Inserire i dati in un database

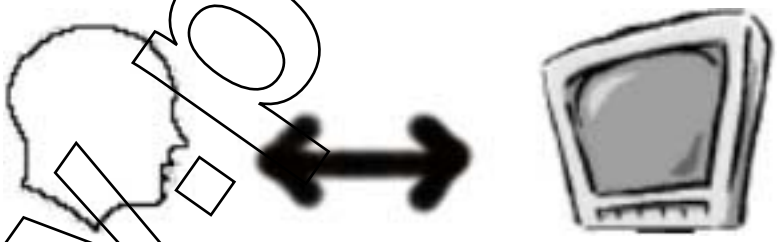

Fig 4: Rappresentazione del rapporto fra file interattivo e utente

Esistono tre livelli di interattività:

- Il primo e semplice livello d’interattività permette all’utente di controllare gli accessi ai contenuti. La TV, per esempio, è

un semplice dispositivo interattivo perché permette all'utente di scegliere i programmi che vuol vedere cambiando canale.

- Un secondo e più complesso livello di interattività permette agli utenti di scegliere il loro percorso attraverso il materiale. Un DVD può essere un esempio di questo secondo livello di interattività, in quanto si accede alle informazioni a richiesta; rispetto alla televisione si può andare avanti, indietro, fermarsi, vedere alcune scene particolari dello stesso film.
- Una terza e più forte forma d'interattività (che oggi rimane sopratutto teorica) permette agli utenti di cambiare gli ambienti lavorando, alterando così non solo la loro esperienza, ma anche l'esperienza degli utenti che seguiranno. A questo livello di interazione, il calcolatore risponde ai bisogni degli utenti su richiesta e forse persino li anticipa. I giochi per computer sono l'esempio più vicino a questo terzo livello d'interattività.

Esistono molteplici modi per favorire l'interazione fra gli utenti e i programmi. Di seguito ne sono esposti alcuni:

- "Command line". L'utente scrive i comandi per il programma, di solito uno per volta. Il programma esegue i comandi e ritorna la risposta, se necessario. MS-DOS e UNIX usano questo stile.
- "Question and answer". L'applicazione pone una domanda e, dopo che l'utente ha fornito una risposta, riporta i risultati. A volte sono chiamate applicazioni "worktrough and use".
- "Menus". Le possibili azioni dell'utente sono messe in una lista sullo schermo e l'utente può scegliere una di queste. Molte applicazioni di MS Windows includono menu.
- "Form filing". L'utente scrive i dati in specifici campi, simile ad un modulo cartaceo in cui sono presenti vari campi da compilare.

- “Function keys”. L’interazione è data da una serie di chiavi speciali o combinazioni di chiavi, per differenti operazioni. Tipici esempi di questa forma di interattività sono i videogiochi.
- “Graphical direct manipulation”. Gli oggetti usati nelle applicazioni sono rappresentati graficamente sullo schermo e l’utente può manipolarli direttamente puntandoli, ciccandoli, trascinandoli, scrivendo, ecc. Molti sistemi a finestre, o GUI (Graphical User Interface) sono basati sulla manipolazione grafica diretta.

5. PRIMO PASSO: COSTRUZIONE DI UN FILE PDF INTERATTIVO

Come prima cosa (dopo aver incluso tra le altre, la libreria Pdflib.h) è stato creato un nuovo oggetto PDF tramite la funzione PDF_new(). Questa funzione restituisce l'handle di un nuovo oggetto PDF, nel caso specifico p.
Successivamente, tramite la funzione PDF_open_file(), è stato creato il file "pagsucc.pdf".
La funzione usata restituisce -1 nel caso in cui il file sia già aperto; questo vuol dire che non si può modificare un documento aperto a run_time; la gestione degli errori è stata sviluppata tramite una istruzione if-then-else che in caso di errore ritornato esce dal programma.

```
// inizio del programma
int
main(void)
{
    PDF *p;             // puntatore ad un oggetto PDF
    int font;           // numero del font usato
    float x, y, z, u;
    int t;              // intero associato alla template
    int b;
    int j;
    int i, k;

// apertura di un nuovo oggetto PDF

    p = PDF_new();
// apertura del file PDF da creare
```

```
if (PDF_open_file(p, "pagseg.pdf") == -1) {
  fprintf(stderr, "Errore: non posso aprire il file PDF\n");
    exit(2);
}
```

Di seguito è stata creata una template; una specie di funzione all'interno della quale ogni operazione di scrittura e di gestione delle immagini viene ridirezionata (invece di agire direttamente sulla pagina attuale) al momento della chiamata. All'interno della template si possono usare tutte le funzioni grafiche di Pdflib, mentre per la gestione delle immagini possono essere usate solo PDF_place_image e PDF_close_image. Non si possono usare le funzioni che gestiscono i collegamenti ipertestuali. Ogni template deve essere definita fuori dalla page description e viene gestita all'interno della page description tramite la funzione PDF_place_image.
All'interno della template, tramite vari cicli while, è stata costruita e colorata la scacchiera; ciò è stato possibile grazie all'utilizzo di funzioni quali PDF_rect(), che costruisce un rettangolo a partire dal punto di coordinate (x, y) e di dimensioni height e widght e Pdf_fill_stroke() che colora l'area disegnata.
Naturalmente per poter definire l'area da colorare bisogna spostare il puntatore utilizzando la funzione PDF_moveto(); allo stesso modo occorre definire ogni volta il colore, da usare per riempire le aree, tramite la funzione PDF_setcolor().
La chiamata di funzione a PDF_end_template definisce la fine della template descritta.

```
// inizio della template
t= PDF_begin_template(p, 600, 850);
// inizializzazione delle variabili
    b=0;
    x=100;
```

```
    y=50;
    z=50;
    u=50;
    i=0;
    k=0;
// creazione e colorazione della scacchiera
while (k<8)
{
        while (i<8)
        {
                if (b%2==0)
                {
// spostamento del puntatore
                        PDF_moveto(p, x,y);
// colorazione dell'area
                        PDF_fill_stroke(p);
// definizione del colore usato
                        PDF_setcolor(p, "fill", "cmyk", 0, 1, 1, 0);
                }
                  else
                {
                        PDF_moveto(p, x,y);
                        PDF_fill_stroke(p);
                        PDF_setcolor(p, "fill", "cmyk", 1, 0, 0, 0);
                }
// disegno di un quadrato della scacchiera
                PDF_rect(p, x, y, z, u);
                b=b+1;
                x=x+50;
                i=i+1;
        }
        i=0;
        b=b+1;
```

```
        x=100;
        y=y+50;
        k=k+1;
}
if (b%2==0)
{
        PDF_moveto(p, x,y);
        PDF_fill_stroke(p);
        PDF_setcolor(p, "fill", "cmyk", 0, 1, 1, 0);
}
else
{
        PDF_moveto(p, x,y);
        PDF_fill_stroke(p);
        PDF_setcolor(p, "fill", "cmyk", 1, 0, 0, 0);
}
PDF_end_template(p);      // fine della template
```

Dopo aver invocato la template, sono state inserite le informazioni riguardanti il documento creato grazie alla funzione PDF_set_info. Essa è stata richiamata più volte modificandone la chiave per definire: il file che da origine al documento PDF, l'autore e il titolo dello stesso.

```
// inserimento delle informazioni associate al file PDF

   PDF_set_info(p, "Creator", "pagseg.c");
   PDF_set_info(p, "Author", "Andrea Gentilini");
   PDF_set_info(p, "Title", "Pagine seguenti");
```

In seguito è stato definito un ciclo for-to-do che ad ogni iterazione crea una nuova pagina del documento tramite l'utilizzo della funzione PDF_open_page; viene colorato lo sfondo della stessa;

viene richiamata la template che disegna la scacchiera e viene invocata la funzione “pagina” passando come argomenti il puntatore all’oggetto PDF(p) e il numero della pagina (i). PDF_close_page infine chiude la definizione della pagina.

```
// ciclo che crea 8 pagine in modalità a tutto schermo, colora lo
// sfondo, richiama la template e la funzione pagina

for (i=1; i<9; i++)
{
	// apertura di una nuova pagina
	PDF_begin_page(p, a4_height,a4_width);
	PDF_set_parameter(p, "openmode", "fullscreen");
	PDF_setcolor(p, "fill", "cmyk", 1, 0, 0, 0);
	PDF_rect(p, 0, 0, 850, 600);
	PDF_fill(p);
	// inserimento della template
	PDF_place_image(p, t, (float) 0.0, (float) 0.0, (float) 1.0);
	// invocazione della funzione pagina
	j=pagina(p, i);
	PDF_end_page(p);		// chiusura della pagina
}
```

La funzione “pagina” ha il compito di disporre i pezzi degli scacchi sopra alla scacchiera, e di definire i link fra le pagine del documento.
All’interno della funzione sono state dichiarate una lunga serie di variabili e puntatori che vengono utilizzati per l’apertura delle immagini raffiguranti i pezzi degli scacchi. L’apertura delle immagini viene fatta attraverso la funzione PDF_open_image che associa a ogni immagine un numero intero; se la funzione ritorna –1 allora l’apertura non è andata a buon fine e il programma termina segnalando una situazione di errore.

```
// dichiarazione delle variabili e dei puntatori
int pb;
int pn;
....
char *pedb = "../../test/wP.png";
char *pedn = "../../test/bP.png";
.....

// apertura delle immagini e associazione delle stesse a degli interi
pb = PDF_open_image_file(e, "png", pedb, "", 0);
   if (pb == -1) {
            fprintf(stderr, "Errore: non apro pedone bianco\n");
            exit(3);
   }
pn = PDF_open_image_file(e, "png", pedn, "", 0);
   if (pn == -1) {
            fprintf(stderr, "Errore: non apro pedone nero\n");
            exit(3);
   }
```

Per rendere più comprensibile il codice scritto all'interno della funzione sono stati posti 8 comandi condizionali (if-then) ognuno dei quali si occupa di definire la struttura interna di ognuna delle 8 pagine da comporre.
All'interno dei vari comandi condizionali sono stati posti due cicli for-to-do incapsulati, che hanno lo scopo di ripercorrere tutte le caselle della scacchiera; il comando switch-case all'interno del ciclo for-to-do più interno ha lo scopo di individuare le caselle della scacchiera nelle quali vanno raffigurati i vari pezzi degli scacchi. Tramite la funzione PDF_place_image i pezzi vengono disegnati nel posto voluto.

Sempre all'interno dei vari comandi condizionali, ogni pagina costruita, viene collegata con delle altre tramite la definizione di link attraverso l'utilizzo della funzione PDF_add_locallink.
Questa funzione prende in input le coordinate di grandezza del link da disegnare e il numero della pagina, appartenente allo stesso documento della invocante, da chiamare.
Per evidenziare meglio i link, sono stati disegnati, nelle zone corrispondenti, dei rettangoli; e, sempre per rendere più leggibile il documento, sono state poste delle scritte tramite le funzioni PDF_set_text_position e PDF_show. Queste due funzioni hanno lo scopo di definire il punto dove porre il testo e di scriverlo.

```
if (num==1)    //creazione della pagina 1
{
    while(b<8)
    {
        io=0;
      y=50;
      j=j+1;
      while(io<8000)
      {
            io=io+1;
            if((io%1000)==0)
            {
//lo switch cerca il quadrato della scacchiera
//sul quale disegnare i pezzi degli scacchi
            switch (io)
            {
            case 1000: if(b==5)
                  {
                  PDF_place_image(e, rb, (float) x+4, (float) y+4,
(float) 1.0);
                  }
```

```
                break;
        case 2000: if(b==4)
                {
                PDF_place_image(e, pb, (float) x+4, (float) y+4,
(float) 1.0);
                }
                break;
        case 4000: if(b==4)
                {
                PDF_place_image(e, cb, (float) x+4, (float) y+4,
(float) 1.0);
                }
                break;
        case 5000: if(b==3)
                {
                PDF_place_image(e, kn, (float) x+4, (float) y+4,
(float) 1.0);
                }
                break;
        case 6000: if(b==0)
                {
                PDF_place_image(e, kb, (float) x+4, (float) y+4,
(float) 1.0);
                }
                else
                {       if(b==2)
                        {
                        PDF_place_image(e, cb, (float) x+4, (float)
y+4, (float) 1.0);
                        }
                        else
                        {       if(b==4)
                                {
```

```
                                PDF_place_image(e, pn, (float) x+4,
(float) y+4, (float) 1.0);
                                }
                            }
                        }
                        break;
                default: break;
                }
                  j=j+1;
                  y=y+50;
                 io=io+1;
                }
            }
    x=x+50;
    b=b+1;
    }
// definizione del font usato
font = PDF_findfont(e, "Helvetica-Bold", "host", 0);
// impostazione della grandezza del font
PDF_setfont(e, font, 30);
PDF_setcolor(e, "fill", "cmyk", 0, 1, 1, 0);
// rettangolo dei commenti
PDF_rect(e, 0, 500, 850, 70);
// rettangolo che ha il compito di evidenziare il link
PDF_rect(e, 550, 250, 250, 100);
PDF_fill_stroke(e);
PDF_setcolor(e, "fill", "cmyk", 0, 0, 0, 1);
// definizione della posizione nella quale scrivere
PDF_set_text_pos(e, 150, 525);
// scrive il testo
PDF_show(e, "IL BIANCO MUOVE E VINCE IN 2 MOSSE");
PDF_set_text_pos(e, 575, 285);
PDF_show(e, "SOLUZIONE");
```

```
// definizione di un link alla pagina 2
PDF_add_locallink(e, 550, 250, 800, 350, num+1, "fitpage");
}
```

Tornando al programma principale, uscendo dal ciclo for-to-do, non rimane che chiudere il documento creato utilizzando la funzione PDF_close() e cancellare l'oggetto PDF creato con la funzione PDF_delete().

```
PDF_close(p);          /* chiusura del documento PDF    */
PDF_delete(p);         /* cancellazione dell'oggetto PDF  */
```

5.1. Output del file “PAGSUCC.PDF”

Di seguito vengono presentate alcune immagini raffiguranti l’output del file “pagsucc.pdf”.
La prima immagine presenta la videata iniziale che compare aprendo il file.
La pagina è composta sostanzialmente da tre parti: in alto è raffigurata in rosso la linea dei commenti al gioco che viene presentato; appena sotto, sulla sinistra compare la scacchiera raffigurante il problema da risolvere, mentre sulla destra è presente il link alle pagine successive contenenti la soluzione.

fig 5: Prima pagina del documento “pagsucc.pdf”

Successivamente viene presentata la seconda pagina del documento creato. Anche essa è suddivisa in tre parti come la precedente, ma sul lato destro della pagina sono presenti tre link a tre pagine differenti che contengono le possibili soluzioni al problema scacchistico presentato. A seconda del percorso scelto vengono aperte la terza, la quinta o la settima pagina del documento.

fig 6: Seconda pagina del documento "pagsucc.pdf"

A seconda del link scelto al passo precedente esistono tre pagine differenti che mostrano la situazione di "scacco matto"; di seguito se ne mostra una. Ovviamente tutte e tre le pagine che visualizzano la fine della soluzione al problema contengono un link che rimanda alla prima pagina del documento.

fig 7: “Scacco matto”, ciccando sul link si torna all’inizio del documento, cioè alla pagina 1

6. SECONDO PASSO: COSTRUZIONE DI UN FILE PDF ATTIVO

Partendo dal codice sorgente del primo file mostrato, è stato creato un documento PDF attivo.
L'idea è di riscrivere all'interno dello stesso documento, e della stessa pagina, tutte le pagine create al primo passo.
Ciò è stato possibile tramite la definizione di una funzione (perdi_tempo) e la modifica di alcune parti del codice precedente.
Rispetto al codice mostrato nel primo passo le funzioni open/close page sono state spostate fuori dal ciclo for-to-do in modo tale da riscrivere sempre all'interno della stessa pagina l'output da visualizzare.

```
PDF_begin_page(p, a4_height,a4_width);      // apre la pagina
for (i=1; i<12; i++)
{
        PDF_setcolor(p, "fill", "cmyk", 1, 0, 0, 0);
        PDF_rect(p, 0, 0, 850, 600);
        PDF_fill(p);
        PDF_place_image(p, t, (float) 0.0, (float) 0.0, (float) 1.0);
        j=pagina(p, i);
        perdi_tempo(p);
}
PDF_end_page(p);        // chiude la pagina
```

La funzione "pagina" è stata modificata togliendo i link che permettevano di viaggiare all'interno del documento, e aggiungendo altre tre pagine che permettono una migliore comprensione del documento creato. Ogni volta che viene

invocata la funzione “pagina”, viene disegnato al posto dei link, un segnatempo composto da 8 rettangoli neri.
In pratica, ogni volta che la funzione scandisce le colonne della scacchiera per cercare il quadrato in cui andare a disegnare l’immagine del pezzo degli scacchi, sulla destra della pagina viene visualizzato un rettangolo nero all’altezza di ogni riga.

```
// questi comandi sono posti dentro il ciclo che scandisce le righe
// della scacchiera

// setta il colore attuale a nero
PDF_setcolor(e, "fill", "cmyk", 0,0,0,1);
PDF_rect (e, 600, y, 200, 50);          // disegna il rettangolo
PDF_fill_stroke(e);                     // colora l’area
```

La funzione “perdi_tempo” definisce un ciclo abbastanza lungo da eseguire, ma non infinito, che riscrive alcune parti della pagina aperta. Nell'esempio disegna sul segnatempo creato dalla funzione “pagina”, dei rettangoli rossi. A ogni iterazione del ciclo viene ridisegnato un rettangolo rosso nella stessa posizione; ogni 10000 iterazioni del ciclo vengono cambiate le coodinate del rettangolo da disegnare spostandolo sopra di una riga.

```
// funzione che ha lo scopo di rallentare l’esecuzione del
// programma

void perdi_tempo(PDF *e)
{
    int k;          // contatore per il ciclo while
    float h;        //  punto in cui bisogna disegnare il rettangolo
    k=0;
    h=50;
    while (k!=80000)
```

```
        {
                // setta il colore da usare
                PDF_setcolor(e, "fill", "cmyk", 0, 1, 1, 0);
                // disegna il rettangolo
                PDF_rect(e, 600, h, 200, 50);
                // colora il rettangolo
                PDF_fill_stroke(e);
                if(k%10000==0)
                {
                        if(k!=0)
                        {
                                h=h+50;       // variazione della coordinata
                        }
                }
                k=k+2;
        }
}
```

Inoltre è stata creata una funzione “sc_matto” che viene richiamata ogni qual volta si è in una situazione di scacco matto.
La funzione richiama “perdi_tempo”, ed è composta a sua volta da un ciclo while che ha lo scopo di rallentare l’esecuzione del programma e visualizza la scritta “scacco matto”.

```
void sc_matto(PDF *e)
{
        int font;
        int io;
        float y;
        perdi_tempo(e);
        font = PDF_findfont(e, "Helvetica-Bold", "host", 0);
        PDF_setfont(e, font, 30);
        PDF_setcolor(e, "fill", "cmyk", 0, 1, 1, 0);
```

```
        PDF_rect(e, 0, 500, 850, 70);
        PDF_fill_stroke(e);
        PDF_setcolor(e, "fill", "cmyk", 0, 0, 0, 1);
        PDF_set_text_pos(e, 200, 525);
        PDF_show(e, "SCACCO MATTO !!!");
        io=0;
        y=50;
// perde tempo ridisegnando sopra al contatore dei rettangoli gialli
// per segnalare la situazione di scacco matto
        while(io<80000)
        {
                io=io+5;
                PDF_setcolor(e, "fill", "cmyk", 0,0,1,0);
                PDF_rect (e, 600, y, 200, 50);
                PDF_fill_stroke(e);
                if((io%10000)==0)
                { y=y+50; }
        }

}
```

E’ importante notare che il documento PDF creato, finchè non arriva a mostrare su schermo l’ultima videata, non mostra alcuna “miniatura” nello spazio a sinistra di Acrobat Reader. Questo perché il programma riscrive ogni volta all’interno della stessa pagina le videate da mostrare; quando il documento avrà finito di mostrare tutte le videate, la miniatura rappresentante l’ultima videata comparirà. In sostanza questo documento crea una sola pagina, quella finale.

6.1. Output del file “SUC_TEMP.PDF”

Di seguito presento l’output del file “suc_temp.pdf” così come si presenta all’utente.
La prima immagine proposta rappresenta la prima videata che compare sullo schermo quando viene aperto il file: come si vede, la struttura della pagina è sostanzialmente uguale a quella della prima pagina del documento descritto al primo passo. L’unica differenza è data dal segnatempo presente sulla destra della pagina, che rimpiazza i link presenti nel documento descritto precedentemente.

Fig 8. Prima videata del documento “pag_seg.pdf”

Acrobat Reader quando viene lanciato per aprire un documento in formato PDF ha, tra le sue opzioni, quella di mostrare sulla destra della videata le miniature, ossia una serie di piccole icone ognuna delle quali è riferita ad una pagina del documento. Dal momento che il file creato è attivo ed ha lo scopo di mostrare tutto l'esempio (il gioco degli scacchi) in una sola pagina, riscrivendola dopo un po' di tempo, inevitabilmente la miniatura viene creata alla fine di tutta la presentazione delle mosse, quindi non è presente durante lo slide-show.

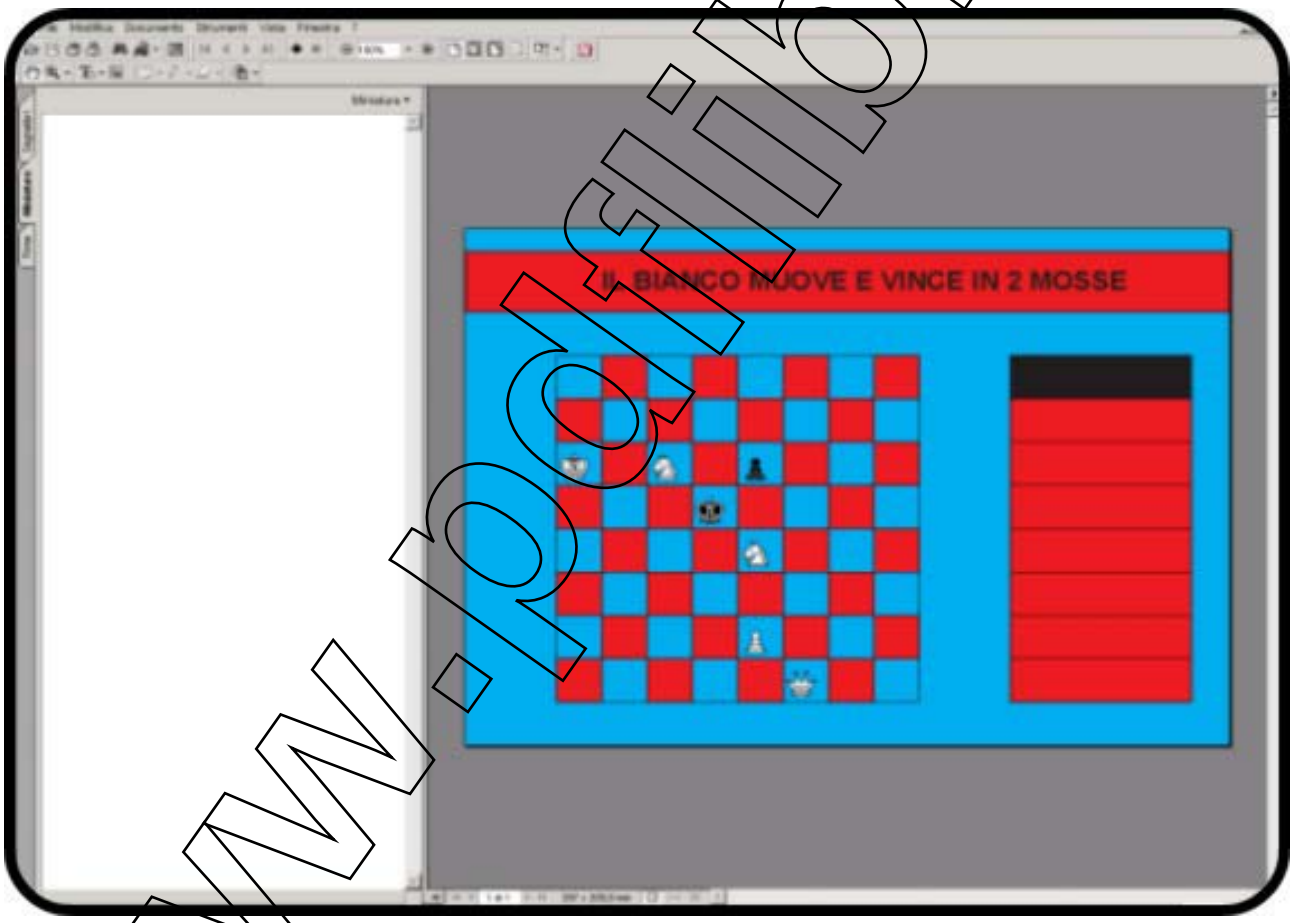

fig 9: Prima videata del file "pag_seg.pdf", con miniature

Rispetto al documento creato al passo 1, in questo documento sono caricate tre pagine in più che evidenziano le tre possibili

ipotesi di soluzione al problema degli scacchi. Esse sono state inserite con lo scopo di rendere più comprensibile lo slide-show.

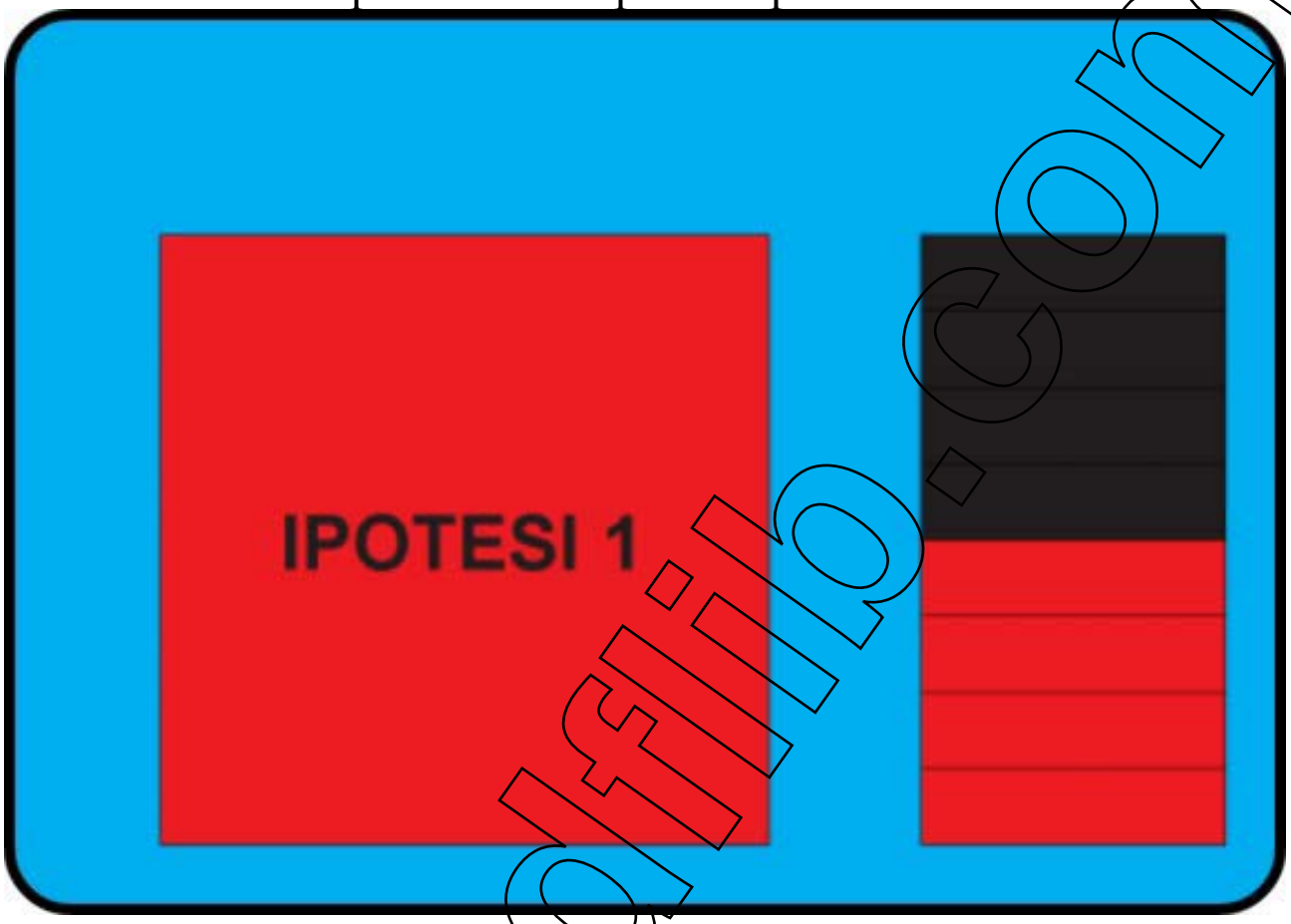

fig 10: Una delle tre pagine che rendono più comprensibile la soluzione del problema all'utente

Ogni volta che il file presenta una situazione di "scacco matto", viene richiamata un'ulteriore funzione che ha lo scopo di rallentare maggiormente il susseguirsi delle pagine. All'utente la situazione di "scacco matto" viene segnalata tramite la colorazione del segnatempo con il colore giallo.

fig 11: “Scacco matto”, sulla destra il segnatempo viene colorato di giallo.

Alla fine dell’esecuzione del documento, l’ultima pagina presentata contiene un link che rimanda all’inizio del documento stesso. La miniatura viene finalmente creata, essa conterrà l’immagine dell’ultima e unica (per il programma) pagina presente all’interno del documento.

Fig 12: Ultima schermata del file “succ_temp.pdf”

7. TERZO PASSO: COSTRUZIONE DI UN FILE PDF INTERATTIVO ASSOCIATO A UN FILE ESEGUIBILE

Partendo di nuovo dal primo passo, tramite la gestione degli errori è stato creato un nuovo file PDF interattivo.
Mentre nel codice sorgente del primo file, se il programma che genera il file PDF veniva lanciato con il file PDF aperto, esso usciva riportando segnalazione di errore; in questo caso se il file dell'output (problema.pdf) è aperto, il resto del documento PDF viene riportato in un altro file (soluzione.pdf).
Per comprendere meglio, riporto sotto il codice del primo programma:

```
// apertura del file PDF da creare

   if (PDF_open_file(p, "pagseg.pdf") == -1) {
       fprintf(stderr, "Errore: non posso aprire il file PDF\n");
       exit(2);
    }
```

Mentre questo è il codice del nuovo programma creato:

```
if (PDF_open_file(p, "problema.pdf") == -1)
       {
           var=1;
           PDF_open_file(p, "soluzione.pdf");
       }
```

Quindi se l'utente, dopo aver aperto il file pdf creato (problema.pdf) esegue nuovamente il programma, il risultato dello stesso viene posto in un'altro file (soluzione.pdf).

Tramite la gestione di una variabile (var) si riesce a creare due documenti PDF diversi. La prima volta che il programma viene eseguito la variabile var è settata a 0, viene aperto il file d'output "problema.pdf" e completato il documento.

```
// p è riferito al file di output "problema.pdf"
if(var==0)
        {
            PDF_begin_page(p, a4_height,a4_width);
            PDF_set_parameter(p, "openmode", "fullscreen");
            PDF_setcolor(p, "fill", "cmyk", 1, 0, 0, 0);
            PDF_rect(p, 0, 0, 850, 600);
            PDF_fill(p);
            PDF_place_image(p, t, (float) 0.0, (float) 0.0, (float) 1.0);
            j=pagina(p, 1);
            ......
            ......
```

Nel caso in cui si esegua il programma nuovamente con il file problema.pdf aperto, la variabile var viene settata a 1 e l'output viene visualizzato nel file soluzione.pdf.
In quest'ultimo caso il file generato (soluzione.pdf) contiene le restanti pagine del documento iniziale mostrato al passo 1.

```
// p è riferito al file di output "soluzione.pdf"
        else
        {
          for (i=2; i<9; i++)
            {
            PDF_begin_page(p, a4_height,a4_width);
            PDF_set_parameter(p, "openmode", "fullscreen");
            PDF_setcolor(p, "fill", "cmyk", 1, 0, 0, 0);
            PDF_rect(p, 0, 0, 850, 600);
```

```
PDF_fill(p);
PDF_place_image(p, t, (float) 0.0, (float) 0.0, (float) 1.0);
j=pagina(p, i);
.....
.....
```

Per rendere interattivi i due documenti creati sono stati definiti dei link fra di loro.
Nel file “problema.pdf” sono stati definiti due link; il primo che ha lo scopo di eseguire nuovamente il file “carica.exe” che quindi creerà il file “soluzione.pdf”, il secondo che collega il documento aperto alla prima pagina del documento “soluzione.pdf”.

```
// link all’interno di “problema.pdf”

PDF_setcolor(p, "fill", "cmyk", 0, 0, 0, 1);
PDF_set_text_pos(p, 590, 335);
PDF_show(p, "CARICA");
PDF_set_text_pos(p, 570, 295);
PDF_show(p,  "SOLUZIONE");
PDF_set_text_pos(p, 610, 145);
PDF_show(p, "APRI");
PDF_set_text_pos(p, 570, 100);
PDF_show(p, "SOLUZIONE");
// apre la prima pagina del file “soluzione.pdf”
PDF_add_pdflink(p, 550, 50, 770, 200, "soluzione.pdf", 1, "fitpage");
// lancia l’eseguibile “carica.exe” che genera il file “soluzione.pdf”
PDF_add_launchlink(p, 550, 250, 770, 400, "carica.exe");
```

Nel file “soluzione.pdf” oltre ai link fra le varie pagine del documento, è presente in ogni pagina un link che rimanda al file “problema.pdf”

// link all’interno di ogni pagina del file “soluzione.pdf”

```
PDF_setcolor(p, "fill", "cmyk", 0, 0, 0, 1);
PDF_set_text_pos(p, 105, 15);
PDF_show(p, " TORNA ALLA PAGINA INIZIALE ");
PDF_add_pdflink(p, 100, 5, 500, 45, "problema.pdf", 1,
"fitpage");
```

7.1. Output dei file “PROBLEMA.PDF” e “SOLUZIONE.PDF”

La prima immagine mostra la prima (e unica) pagina del documento “problema.pdf”.
La pagina è strutturata nella stessa maniera delle pagine dei documenti presentati nei due passi precedenti.
Questa volta sulla destra della pagina sono presenti due link.
Il primo link (ndr. “carica”) lancia l’esecuzione del file “carica.exe” che costruirà il file “soluzione.pdf”.
Il secondo link apre la prima pagina del documento precedentemente creato.

fig 13: Prima pagina del file “problema.pdf”

Il file “soluzione.pdf” si presenterà all’utente in questo modo:

fig 14: Prima pagina del documento "soluzione.pdf"

In questa pagina ci sono 4 link; i tre posti sulla sinistra consentono all'utente di viaggiare interattivamente all'interno dello stesso documento ("soluzione.pdf"), mentre quello posto sotto la scacchiera consente all'utente di tornare al file di partenza ("problema.pdf").

8. CONCLUSIONI

Il lavoro illustrato in questa tesi, mostra come si può efficacemente usare un linguaggio di programmazione comune, quale è il C, per poter programmare dei file PDF attivi e interattivi. Tuttavia, occorre verificare i tre documenti proposti dal punto di vista della portabilità, essendo questa la proprietà più importante e la caratteristica fondamentale del formato PDF.
Per dare un giudizio, i tre documenti creati, sono stati testati su elaboratori con differenti sistemi operativi installati. I primi due file creati "pagsucc.pdf" e "pag_seg.pdf" funzionano correttamente su tutti e tre i sistemi operativi testati (Windows, Unix, MacOs). Il terzo file ("problema.pdf") viene perfettamente aperto da tutti e tre i sistemi operativi, ma il link che carica il file eseguibile ("carica.exe") funziona solo in ambiente Windows, in quanto il programma è stato compilato e eseguito nell'ambiente stesso. L'esecuzione del file negli altri due ambienti, Unix e MacOs, non porta alla creazione del file "soluzione.pdf". Tuttavia, una volta creato, il documento "soluzione.pdf", viene regolarmente aperto e linkato anche negli altri sistemi operativi.

I risultati ottenuti da questo lavoro di tesi sono importanti, essi sono destinati a porre le basi per un futuro sviluppo di documenti PDF interattivi migliori.
Sicuramente uno sviluppo maggiore di questo tipo di documenti è favorito dalla possibilità di poter programmare attraverso l'utilizzo di più linguaggi di programmazione e in molteplici ambienti di lavoro, supportati dalla libreria PDFlib.
Per quanto riguarda l'aspetto estetico e puramente gestionale di partite a scacchi, si dovrebbe cercare di gestire le partite attraverso l'utilizzo dei "font degli scacchi".

Per quanto riguarda l'aspetto interattivo dei documenti creati, sarebbe interessante poter trasferire dei dati attraverso l'utilizzo di documenti PDF. In altre parole, cercare di stabilire un rapporto client-server tra documenti PDF, magari attraverso l'implementazione di form, aumenterebbe enormemente la capacità di interazione dei documenti.
PDFlib, tuttavia ha alcune limitazioni che al momento non favoriscono questo tipo di approccio; in particolare, le funzioni che creano dei link all'interno dei documenti, non restituiscono nessun valore, infatti ritornano "void".
Tuttavia esistono varie librerie che forniscono la possibilità di programmare dinamicamente file PDF; per Java ad esempio, ci sono "Bigfaceless" e "Itext".

9. BIBLIOGRAFIA

[1] Michael Yacci: “Interactivity Demystified: A Structural Definition for Distance Education and Intelligent CBT”, Rochester Institute of Technology, June 2000

[2] Partha Dasgupta (Department of Computer Science, Arizona State University), Ayal Itzkovitz and Vijay Karamcheti (Department of Computer Science New York University): “Active Files: A Mechanism for Integrating Legacy Applications into Distributed Systems”, January 2000

[3] C. Scott Miller: “Adobe's Portable Document Format: Document Standard for Publishing's Future”, August 1999

[4] Vincent Ribière: “ Introduction to Multimedia, Creativity & Computers”, CSIS Dept. American University, September 2001

[5] A. Camurri, R. Trocca, G. Volpe: “Interactive Systems Design: a KANSEI-based approach”, DIST - University of Genoa, April 2002

[6] L. Bompani, P. Ciancarini, F. Vitali: “Active Documents in XML”, Dept. of Computer Science, University of Bologna, May 1999

[7] M. Vazirgiannis, D. Tsirikos, Th. Markousis, M. Trafalis, Y. Stamati, M. Hatzopoulos, T. Sellis: “Interactive Multimedia Documents: a Modeling, Authoring and Rendering approach”, June 1999

[8] Adobe systems incorporated: “PDF Reference 3rd edition, version 1.4”, Addison-wesley 2001

[9] Michalis Vazirgiannis: “Interactive Multimedia Documents Modeling, Authoring, and Implementation Experiences”, 2000

[10] Thomas Merz: “PDFlib: a library for generating PDF on the fly. Reference Manual, Version 4.0.3”, June 2002

[11] J. V. Dempsey, M.P. Driscoll: “Interactive Instruction and Feedback”, NJ Educational Technology Publications, 1993

[12] M.G. Moore: “Three types of interaction”, The American Journal of Distance Education, 1989
[13] Thomas Merz: “Web publishing with acrobat/PDF”, New York, 1998
[14] N. Hirzalla, B. Falchuk, A. Karmouch: “A Temporal Model for
Interactive Multimedia Scenarios”, 1995
[15] Adobe systems incorporated: “PDF as a standard for archiving”, San Jose, 2002
[16] H: Hagen: “Integratine TEX, MetaPost, JavaScript and PDF”, Spring 1998

Link a siti sul formato PDF

http://www.adobe.com/type/
http://www.pragma-ade.com/
http://www.document-solutions.com/services_pdfen_nav.htm
http://www.pubserv.washington.edu/forms/search/forms_pdf.html
http://www.adobe.com/products/acrobat/main.html
http://www.planetpdf.com
http://www.pdfzone.com

Link a siti scacchistici

http://www.enpassant.dk/chess/diaeng.htm
http://users.libero.it/rigel_g/
http://sourceforge.net/projects/jcheesboard/
http://www.enpassant.dk/chees/diaeng.htm
http://arska.org/goodmarks/?hauva