Department of Informatics
University of Fribourg (Switzerland)

PHYSICAL AND LOGICAL STRUCTURE RECOGNITION OF PDF DOCUMENTS

THESIS

presented to the Faculty of Science of the University of Fribourg (Switzerland)
in consideration for the award of the academic grade of *Doctor scientiarum informaticarum*

by

Jean-Luc Bloechle

from

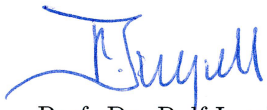
Auswil BE (Switzerland)

Thesis N° 1676
UniPrint, Fribourg
2010

Accepted by the Faculty of Science of the University of Fribourg (Switzerland) upon the recommendation of Prof. Dr. Thierry Paquet, Université de Rouen, France and Prof. Dr. Salvatore-Antoine Tabbone, Université de Nancy 2, France.

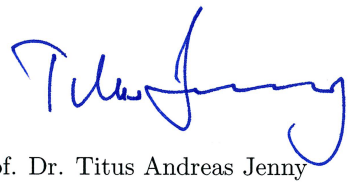
Fribourg, June 25^{th}, 2010

Thesis supervisor



Prof. Dr. Rolf Ingold

Faculty Dean



Prof. Dr. Titus Andreas Jenny

Acknowledgments

"Against the tide, **we** finally reached the coast."

Yes, this thesis is both a personal and collective achievement. For sure, I would never be able to cruise during all these years and reach the coast alone. Our ship needed a well trained crew; I, in the role of the helmsman, my thesis director, in the role of the captain, and of course all my fellow sailors. And now, it is my duty to show you my gratefullness, dear fellow sailors !

THANK YOU ALL

Yes, thank you all, my thesis director, my working collegues and friends, my family, my love, and of course my adorable cat. Special thanks go to: Rolf Ingold, my thesis director, who showed me the path to follow, Denis Lalanne and Maurizio Rigamonti who carefully proofread my thesis, Dominique Bloechle who improved my English, and last but not least, Virginie Fornerod who was there for me every time I needed her.

Abstract

Physical and logical structure recovering from electronic documents is still an open issue. In practice, there are tremendous needs for recovering the underlying structures of existing electronic documents. Such information allow documents to be reedited, restyled, or reflowed; additionally, document structures greatly improve indexing and retrieving. For instance, accessibility, a key feature of electronic documents, precisely needs structured electronic content. Indeed, it is of highest importance to ensure documents access to screen readers and other adaptive equipments used by disabled people, such as synthetic speech or Braille ouptut systems. In a similar way, portable devices such as PDA - personal digital assistant - and smartphones such as Apple's iPhone and Google's Nexus One phone need electronic documents to be adapted and reflowed in order to fit their low screen resolution.

As a consequence, structure recovery from PDF documents is essential, since practically every printable document can be converted into the PDF file format without much effort. However, PDF has primarily been designed for the preservation of the layout instead of for further editing. So, when a word processor, spreadsheet, or presentation application exports to a PDF, the document's layout is guaranteed, whereas its underlying structures are either partially or even completely lost. In broad terms, users must keep in mind that creating a PDF is not a reversible process.

In this thesis we present a restructuring workflow that precisely tackle the poor reversibility issue of PDF files. We first detail the specifications of the PDF file format, its major features together with its strengths and weaknesses. Then, we report about our evaluation of different existing tools for PDF content extraction and analysis. We also study state-of-the art researches about document analysis, and focus on the recovering of physical and logical structures from electronic documents. In order to fill in the gap between users needs and existing systems, we propose a flexible and efficient approach for recovering document structures from PDF files. Our strategy is applicable to every kind of composite electronic documents containing text, images, and graphics; it is built upon a two-stages process. The first phase recovers the physical structure and generates an intermediate XML-based canonical format. The second phase is based on the canonical format and aims at recovering logical

structures from the physical one by inferring document models thanks to a dynamic user interface providing interactive and incremental learning functionalities.

Our work brings various contributions to the document analysis community. For instance, the recognition of the physical structure of a PDF document is entirely based on its electronic content, i.e., the physical structure is recovered by analyzing only the electronic primitives. The physical structure recognition algorithm is fully automatic because it uses a set of dynamic features (relative to the processed content), it also brings new restructuring methods to the document analysis community. The output of the physical restructuring process is stored together with the document content and layout in an original XML-based canonical document format. Eventually, our investigations led to an innovative document analysis architecture, based on the canonical document format, which enables document logical structures to be progressively inferred and integrated in a knowledge base thanks to an interactive and dynamic learning environment. Concrete experiments and evaluations show that our complete "reverse engineering" workflow is efficient, and, as a consequence, brings a strong contribution to the document analysis community.

Résumé

La reconnaissance des structures physiques et logiques de documents numériques est un thème de recherche d'actualité. Il existe une réelle demande quant à la reconnaissance de telles structures pour des documents numériques de toutes sortes. En effet, l'information structurelle offre la possibilité de rééditer, reformater, ou réutiliser le contenu des documents numériques; elle permet également d'améliorer leur indexation et donc d'affiner leur recherche. De nos jours, l'accessibilité est une caractéristique essentielle des documents numériques. D'une part l'information doit être accessible aux personnes handicapées par le biais de lecteurs d'écran: une aide technique utilisant le contenu et la structure d'un document pour le retranscrire par synthèse vocale et/ou en braille. D'autre part, les récents progrès technologiques apportent une offre considérable en dispositifs numériques portables pour lesquels le contenu doit également être adapté et reformaté afin d'être affiché correctement sur des écrans basse résolution; on peut citer en particulier, les PDAs (assistants numériques personnels) et autre téléphones portables tels que l'iPhone de Apple et le Nexus One de Google.

Dans ce contexte, l'extraction des structures de documents PDF est d'une importance capitale, puisque tout document numérique imprimable peut être converti au format PDF. Le format de fichier PDF a en effet été conçu pour préserver fidèlement l'apparence d'un document, cela au détriment de sa structure sous-jacente. Cette caractéristique affecte particulièrement la réutilisation des documents PDF qui ne sont dès lors pas créés dans l'optique d'être accessibles (en toute généralité). Ainsi lorsqu'une application d'édition de texte ou de présentation exporte au format PDF, la mise en page du document est assurée, tandis que les structure physiques et logiques sont la plupart du temps partiellement, voire totalement, perdues. L'utilisateur doit garder en mémoire que la création d'un document PDF n'est pas un processus réversible.

Cette thèse présente un ensemble d'étapes permettant précisément de résoudre les problèmes liés au manque de structures des documents PDF. En premier lieu, le format PDF lui-même est abordé, sa spécification est détaillée, tandis que ses qualités et défauts sont exposés. Différents outils permettant d'extraire l'information de documents PDF ainsi que de les analyser sont ensuite présentés. Nous étudions ensuite en détail l'état de l'art des recherches con-

cernant l'analyse de document, et plus précisément la reconnaissance des structures physiques et logiques de documents numériques. Notre but est de combler le fossé séparant les besoins des utilisateurs et les systèmes existants. Dans cette optique, nous proposons une approche flexible et efficace pour la reconnaissance des structures de documents PDF. Notre stratégie est basée sur un processus divisé en deux étapes clefs, elle est applicable à toutes sortes de documents numériques composés de texte, d'images, et de graphiques. La première étape extrait la structure physique d'un document, puis génère un format XML intermédiaire appelé "document canonique". La deuxième étape se base sur le document canonique, donc sur la structure physique précédemment extraite, afin de reconnaître la structure logique. Cette étape a pour but d'inférer des modèles de documents au travers d'une interface utilisateur dynamique offrant des fonctionalités d'apprentissage interactif et incrémental.

Nos recherches apportent diverses contributions à la communauté de l'analyse de documents. En particulier, la reconnaissance de la structure physique est uniquement basée sur le contenu numérique des documents PDF. L'algorithme développé pour la reconnaissance de la structure physique décrit de nouvelles étapes de restructuration précieuses dans notre domaine de recherche. De plus, cet algorithme est entièrement automatisé, il bénéficie en effet de seuils dynamiques relatifs au contenu à restructurer. Le résultat de la restructuration physique est quant à lui sauvegardé, tout en conservant le contenu exact ainsi que le formatage du document original, dans un format de fichier XML spécialement développé à cet effet, le format canonique. Nos investigations nous ont finalement conduit au développement d'une architecture d'analyse de documents novatrice basée sur le format canonique. Cette architecture permet à l'utilisateur d'inférer des modèles de documents progressivement et interactivement, au moyen d'un environement d'apprentissage incrémental et dynamique. Un ensemble d'expériences et d'évaluations démontrent que notre processus complet de "rétro-ingénierie" est efficient et que, de ce fait, il apporte une contribution non négligeable au domaine de l'analyse de documents.

Contents

1 Introduction

Contents

In the context of this work, when speaking about electronic documents we mean documents stored in a computer intelligible format and containing mainly static (no time dependence) and textual information.

But prior to tackle the issue of electronic document analysis, this chapter gives an introduction about it. That is, the context of this thesis is first highlighted in Section 1.1. Then, Section 1.2 emphasizes the contribution of our work in the field of document analysis. Finally, Section 1.3 outlines the structure of this report.

1.1 Context

Paper documents are a fundamental communication medium between human beings. Since the beginning of the computer era, electronic documents have gained an increasing role in human-human interactions. Beyond this, existing paper versions of documents are more and more replaced by their electronic form (e.g., google book library project). While individuals are able to instantly read and decipher the meaning of such documents, computers are only good at processing their content, without giving any interpretation. Thus, document analysis tries to fill this gap by recovering high level information and structures (i.e., physical and logical structures) from documents which could drastically improve indexing and information reuse.

Document analysis methods have traditionally been developed for paper documents acquired by scanners or cameras. However, structural document analysis becomes more and more useful when dealing with documents that already exist in electronic form. In practice,

there are tremendous needs for recovering the physical and logical structures of existing electronic documents. For instance, such structures allow documents to be reedited, restyled, or reflowed; additionaly, they also greatly improve their indexing and retrieving.

In this context, structure recovery from PDF documents is of highest importance, since practically every printable document can be converted into PDF without much effort. However, the PDF file format has primarily been designed for the preservation of the layout at the expense of further editing. So when a word processor, spreadsheet, or presentation application exports to a PDF, the document's layout is guaranteed, whereas its structures are either partially or even completely lost. Thus, users must keep in mind that creating a PDF is not a reversible process because of the limitations of PDF as a format.

Research on PDF structure recognition, and more generally, electronic document recognition, has greatly inherited from image document analysis and therefore, existing electronic document analysis systems do not fully benefit of the electronic primitives they contain. Indeed, in the era of electronic documents, most of the recognition system still relays on the image representation of documents leaving their internal electronic content either partially or totally unused. For instance, the font size, the font name, the character position, dimension, and unicode are accurate electronic information that would certainly improve the physical structure recognition of analysis systems. However, accessing the detailed content of PDF document is not easily feasible because of the intricacy of its format specification. PDF tools and programming interface only allow to display accurately documents on screen (or at printing), whereas the internal access of their raw electronic primitives is not available.

Another limitation of current analysis systems resides in the document recognition formats they use. Existing formats (see Section 3.6 "Document Recognition Data Formats", in Chapter 3) only store the content and structures of documents, while leaving their original layout aside. On the opposite, file formats such as PDF preserve the original document appearance, although they do not provide simple and easy ways to store their content and structures.

At a higher level, logical structure recognition is a complex task that can only be achieved thanks to the elaboration of a document model, i.e., a knowledge base inferred from a representative set of annotated documents. In the recent past, some tools for the automatic acquisition of document models have been developed (see Section 3.4 "Document Image Anaysis" and Section 3.5 "Electronic Document Analysis", in Chapter 3), however they are subject to severe limitations. First, current systems only concentrate on predefined sets of documents, thus greatly narrowing their field of application. Second, interactive and dynamic training is absent of existing learning systems, requiring experienced users to manually annotate the ground-truthed data. The annotation of data is a cumbersome task that requires huge amounts of human striving as well as time, and, therefore, is very expensive. Moreover, the characteristics of an annotated corpus are static, whereas those of new documents of a specific class may evolve during time. This means that existing databases need also to

be updated with new data sets in order to match with new potential specifications of the recognition system.

1.2 Contribution

As pointed out in the previous section, research on electronic document recognition has expanded from paper document recognition, therefore exploiting primarily the bitmap image of scanned documents. Thus, none of the existing systems consider the full recognition of electronic document structures. Worst, existing systems only focus on very specific classes of documents; they clearly lack flexibility and do not adapt easily to new document classes. Analysis systems should dynamically support a wider range of document classes. Besides, recognition systems need annotating data to infer their recognition knowledge base, i.e., document model. The availability and the collecting of such corpus is much less an issue compared to the cost of their annotation. There is a need for assisted learning environments with user-friendly interfaces, recognition improving with use, and models being refined incrementally.

Therefore, we propose a generic approach to recover document content, layout, and structures, i.e., physical and logical structures, that solves the aforementioned issues. Our strategy is applicable to every kind of composite electronic documents containing text, images, and graphics; it is built upon a two-stages process (see Figure 1.1). The first phase is handled by a tool called "XED", it recovers the physical structure and generates an intermediate XML-based canonical format called "OCD". The second phase is handled by "Dolores", a tool that aims at recovering logical structures from the physical one by inferring document models thanks to a dynamic user interface providing interactive and incremental learning functionalities.

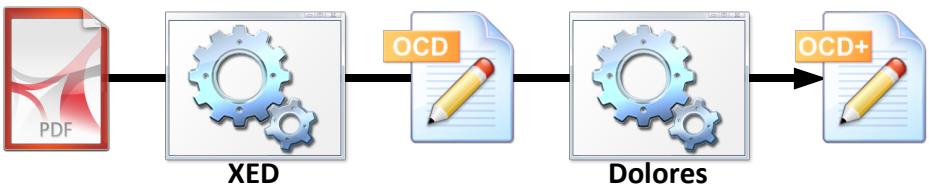


Figure 1.1: Our generic approach to recover document structures is composed of two main modules: the first one, called "XED", recovers the physical structure, whereas the second one, called "Dolores" recognizes the logical structure.

Our work brings various contributions to the document analysis community. For instance, XED, our tool that recognizes the physical structure of PDF documents, integrates some interesting innovations. First, it is based on the electronic content of documents and therefore recovers their physical structure by only analyzing their electronic primitives. Then, the physical structure recognition methodology itself integrates dynamic features (relative to the

processed documents) and brings some new key restructuring steps such as the "retroactive merging" and the "post-linearization".

Concerning the result of the physical restructuring process, it is stored together with the document content and layout in OCD, a dedicated XML-based format. In fact, existing document recognition formats only store the content and structures of documents, while leaving their original layout aside. On the opposite, file formats such as PDF preserve the original document appearance, but they do not provide simple and easy ways to store their content and structures. OCD is our solution, it is a standalone canonical XML format for representing the content, layout, and structures in a unique and non ambiguous manner. That is, the OCD file format preserves the exact appearance of the original PDF document while keeping a clear description of both its content and recovered physical structures. Moreover, the OCD file format optimizes the file size storage by precisely benefitting of its internal structured content, i.e., OCD files composed only of textual content have smaller sizes than PDF ones.

Some fundamental innovations have also been achieved during the elaboration of Dolores, our solution in order to recover the logical structures of electronic documents. For instance, in existing analysis systems, the transformations from physical to logical structures is either hard-coded in some programming language, or inferred by a corpus of annotated data. Such a situation is acceptable for small and midsize applications working with specific document classes whose layouts are simple and close enough. However, in the case of loose or evolving document classes, i.e., the physical structure may have a great variability, more sophisticated analysis tools are required. Considering this, our investigation led to a new type of document analysis architecture that allows document models to be setuped and refined according to user interactions. That is, our system does not need corpus of annotated data anymore since these are precisely inferred progressively and integrated in document models, thanks to the interactive and dynamic learning environment of Dolores. Additionally, the knowledge contained in a document model is revealed to the user thanks to an interactive multilayer perceptron, i.e., a "transparent" learning machine we developped, derived from standard artificial neural networks.

Our experiments showed that XED and Dolores improve drastically the usability and flexibility of document analysis systems. Moreover, our complete "reverse engineering" workflow is greatly facilitated by the use of OCD as a pivot format.

1.3 Thesis Structure

This thesis is organized as follows:

Chapter 2: Document Production and Understanding. The purpose of this chapter is to set the fundamental background about document production and understanding. Some key concepts are therefore introduced and defined: document physical and logical

structures, document analysis and structures recognition, document models.

Chapter 3: State of the Art. A review of the state of the art tools and researches in the field of document analysis is presented. Existing document recognition data formats are also studied.

Chapter 4: The Pros and Cons of PDF Files. An overview of the PDF file format is given, the emphasis is put on its strenghts and weaknesses. A reverse engineering workflow is then presented as a solution to overcome the physical and logical structures inconsistencies of PDF documents.

Chapter 5: Canonical Document Extraction. This chapter presents XED, our physical restructuring tool based on the PDF file format. The restructuring results of XED are stored thanks to a new formalism called "Canonical Document Format" used to express the content, the layout, and the physical structures of electronic document. An innovative physical restructuring methodology is then presented in details.

Chapter 6: XCD & OCD, Two Canonical Document Formats. This chapter presents two XML-based formats we elaborated in order to store canonical documents, resulting from the physical restructuring of PDF documents, in a human and machine readable description. The first format is called XCD - XML Canonical Document - and stores the canonical document as raw XML content. The second format is a strong optimization of the first one, it is called OCD - Optimized Canonical Document - and uses different techniques that greatly reduce the file size storage of a canonical document.

Chapter 7: Dolores, Interactive Logical Restructuring. Dolores - Document Logical Restructuring - is an interactive windowed environment that aims at inferring document models from canonical documents. The chapter presents the interactive and dynamic labeling interface of Dolores, as well as its innovative document model, and learning system.

Chapter 8: Case Studies. In this chapter, five case studies are described. The goal of these concrete experiments and their evaluation is twofold. First, we want to validate the canonical document format through the use of Dolores, and then, we want to demonstrate the relevance of Dolores itself as a system for document logical restructuring.

Chapter 9: Conclusions and Perspectives. Finally, Chapter 9 concludes this thesis with some discussions and perspectives.

Appendix B: Feedforward Artificial Neural Networks - Multilayer Perceptrons. A detailed theory about artificial neural networks, and, more specifically multilayer perceptrons are presented. As a matter of fact, a multilayer perceptron is a restrained artifical neural networks that contains no recursive loop and where the information is always propagated forward through the layers.

2

Document Production and Understanding

Contents

This chapter introduces the main concepts and definitions about document production and understanding. The first section, Section 2.1, defines the notion of printed document in the context of this thesis. Then Section 2.2 stresses the steps and structures involved in the document production, whereas Section 2.3 presents its reversed process, i.e., the document understanding. Finally, Section 2.4 exposes the knowledge structures, called document models, that are needed to achieve the document understanding process.

2.1 Printed Documents

In the context of this thesis, *printed documents* always refers to documents that have been printed to an output, i.e., rendered to a fixed-layout presentation format. More precisely, printed documents include three different levels of document outputs: paper documents (hard copies), raster image documents (bitmaps), and electronic documents (page description languages). A page description language [71] - PDL - is a file format specifying a language that describes the appearance of a printed page (and/or document) in a higher level than an actual output bitmap, e.g., DVI, PDF, PostScript, SVG, or XPS are various existing PDL (see Figure 2.1).

During their rendering process, printed documents generally lose crucial information about their original structures that partially or even fully disables document future reuse

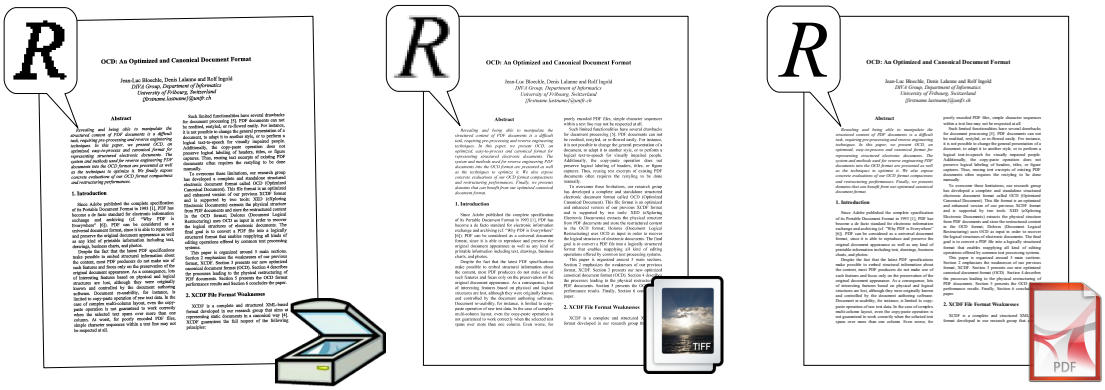


Figure 2.1: Printed Documents may be hard copies (or scanned hard copies, at left), raster image files (at center), and PDL files (such as PDF, at right).

such as manipulation or modification. Therefore, a complete document reverse-engineering phase has to be achieved in order to recover the original document content and structures, thus allowing the document to be used again.

For an individual, accessing the meaning of a printed document involves three levels of perception: vision, recognition, and understanding. The content of a printed document is instantly perceived thanks to the human visual system, but it may also be perceived thanks to other human senses, in order to accomodate visually impaired people. Indeed, the sight sense can be substituted by the hearing sense: a textual document can be translated into an audio stream thanks to a text-to-speech system or simply by using a pre-recorded human voice. The sight sense can also be substituded by the touch sense: the Braille system is a method that is widely used by blind people to read and write textual content. Then, the recognition and understanding of a document are tightly coupled to the sensitive perception system. Actually, all sensitive perceptions are treated by our brain and nervous system in order to transform the input information into a complex and abstract representation.

Whereas human beings are precisely able to efficiently process visual data in order to first recognize them and then understand their meaning, computers do not have such capabilities. Their power mainly resides in their efficieny to manipulate very large amounts of data in a very short time, they are essentially big computing machines. Thereby, their “visual capabilities” are very basic, computers are just able to process raw visual data, i.e., scan, store, transform, print, and render documents.

The aim of this thesis is precisely the recognition of printed documents, and in particular those having *complex structures*, i.e., documents with a rich layout, such as newspapers. This chapter stands as a background for this thesis, it is essential in order to present the context and definitions relative to the recognition of printed documents.

2.2 Document Production

A printed document is the result of a multi-step production process called *document production* (see Figure 2.2). First, the content of a document is edited by a human operator, leading to a *logical document*. Then, various layout rules are applied on the edited content resulting in a formatted document called the *physical document*. And finally, the physical document is rendered to a final fixed-layout document format called the printed document.

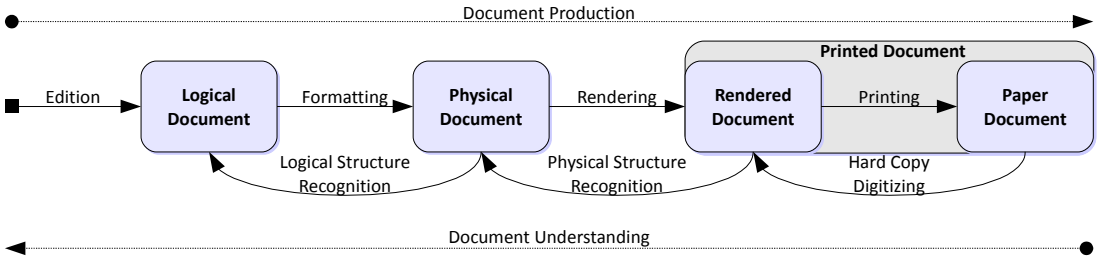


Figure 2.2: The document production and understanding cycles. Our definition of printed document include rendered documents as well as paper documents.

Depending on the editor, the content edition and the formatting of the document may be separated or mixed together. For instance, typesetting systems such as LaTeX [63] (the one used to edit this thesis) precisely enforce the author to separate the content edition from the formatting. Indeed, LaTeX is based on the idea that authors should be able to focus on the content of what they are writing without being distracted by its visual presentation. In preparing a LaTeX document, the author specifies the logical structure using familiar concepts such as chapter, section, table, figure, etc., and lets the LaTeX system worry about the presentation of these structures. On the other way, standard word processors such as Microsoft Word [118], allow the author to edit the document content together with its formatting, which may lead to unconsistent layouts.

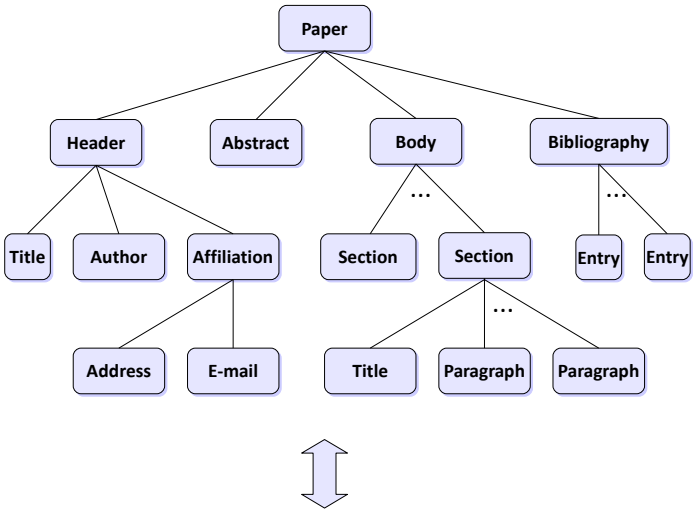
2.2.1 Logical Document

The first step of the document production process consists in editing the content of a document with a text editor. The author of the document organizes its content in term of hierarchical relationships, therefore inferring a *logical structure*. Thus, an authored document is called a logical document.

The logical structure of a document associates semantic information to the physical document. It reflects the way information is organized in terms of logical objects, i.e., chapters, titles, paragraphs, figures, headers, etc. The logical structure specifies the function and meaning of the physical objects forming the document and the relationships between them. More precisely, the relationships between the logical objects in the logical structure are typically in

the form of hierarchichal include and sequence relations. For example, a document includes a title, authors, summary, and a sequence of chapters, each of them including a title and a sequence of sections, and so on.

Thereby, the logical structure is commonly represented as a tree structure in order to transcribe the hierarchial relationships existing between the various logical objects. Figure 2.3 expresses the logical structure of the document shown on Figure 2.4 as a tree structure and formalized in an XML-based representation.




```
<paper>
 <header>
  <title content="OCD: An Optimized and Canonical ..."/>
  <author> ... </author>
  <affiliation> ... </affiliation>
 </header>
 <abstract> ... </abstract>
 <body>
  <section>
   <title content="Introduction"/>
   <paragraph content="Since Adobe published the ..."/>
   ...
  </section>
  ...
 </body>
 <bibliography> ... </bibliography>
</paper>
```

Figure 2.3: The logical structure of a document is commonly represented as a tree structure (at top) itself formalized as an XML description (at bottom).

Various documents may have similar or different logical structures. And thus, a *class of documents* is basically the set of all the documents that can be described with the same logical structure formalism. Nevertheless, various classes of documents contain documents having different logical structures. For example, the logical structure of a legal document is completely different from the logical structure of a business plan or a scientific paper.

2.2.2 Physical Document

In the production process of a document, the transformation from the logical document to the physical document is done thanks to formatting rules. Depending on the editing software, these rules may be more or less implicit. Thus, poor editors allow the user to mix the edition and formatting processes and may lead to non-homogeneous formatting styles (e.g., Microsoft Word, Open Office Writer). On the opposite, structure oriented document preparation system separate formatting data into standalone stylesheets (e.g., LaTeX, . For instance, this enables a single document to be easily declined under various formatting styles or multiple documents to have a similar presentation.

In a physical document, the logical structure is no more explicitly written, but it is revealed thanks to the document's typographical attributes and layout, i.e., *the physical structure*. The physical structure of a document corresponds to the organization of the page in terms of a hierarchy of regions delimited by images, graphics, and text blocks that can be further split into text lines, words, and characters.

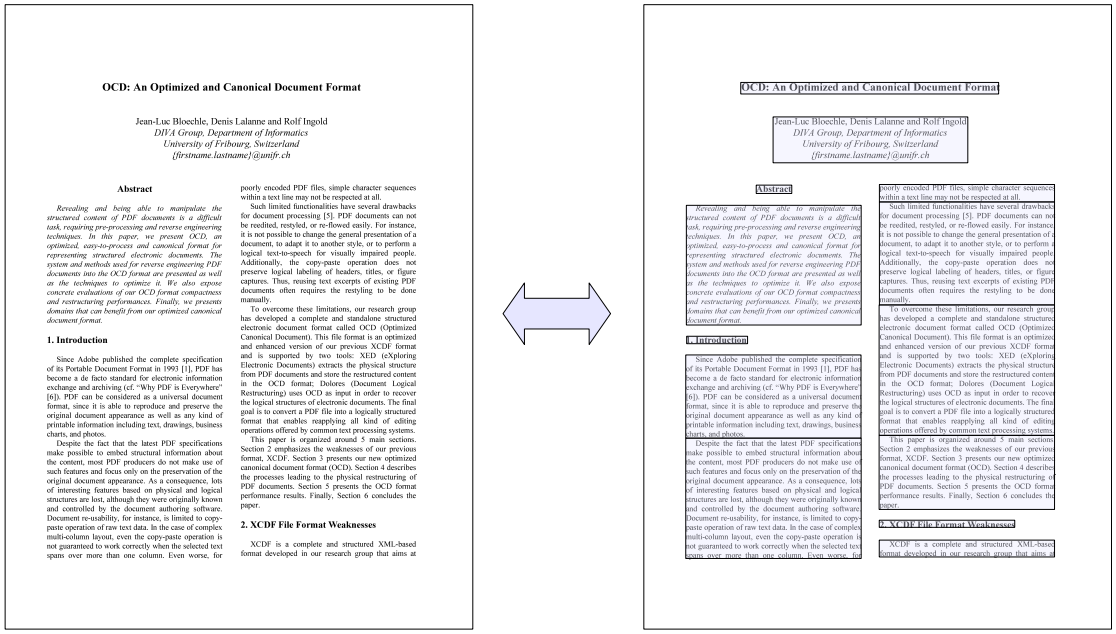


Figure 2.4: On the left, the first page of a conference paper, and on the right, the text blocks segmentation corresponding to the physical structure of the conference paper.

Thus, the visual appearance of a printed document simply reflects its physical structure, without any semantic interpretation. As the logical structure, the physical structure is often represented as a tree structure in order to transcribe the hierarchial relationships existing between the various *physical objects*. Figure 2.5 expresses the physical structure of the document shown on Figure 2.4 as a tree structure and formalized in an XML-based representation

(see “XML-Based Formats” in Chapter 3).

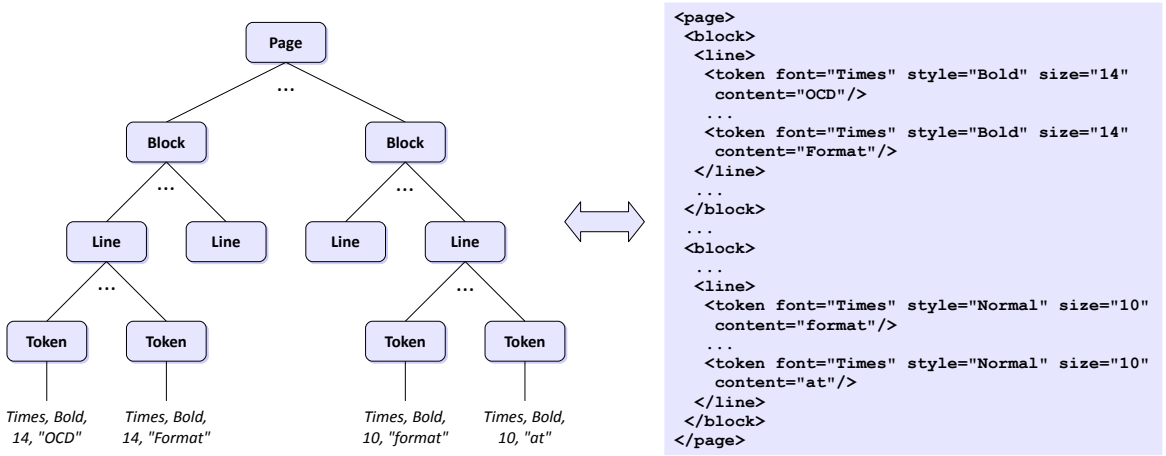


Figure 2.5: The physical structure of a document is commonly represented as a tree structure (on the left) itself formalized as an XML description (on the right).

Whereas all the documents can be described thanks to a universal physical structure formalism, logical structures depend on the targeted applications and are restricted to corresponding document classes.

2.2.3 Rendered and Paper Documents

During the third step of the document production process, the physical description of the document is rendered as an image. In computer sciences, the term “rendering” is commonly used to define the process of drawing a document described with high-level primitives on a raster image, i.e., uniquely composed of pixels. However, in the context of this thesis, the term “rendering” is more generic and expresses the drawing of a document on a fixed-layout format, i.e., a raster image as well as a vectorial image (such as a PDL document).

Finally, during the fourth step of the document production process, the rendered image is outputted on a physical media such as paper or transparencies (hard copy), thanks to a printing device. As exposed in Section 2.1, we use the term printed document to refer both to rendered and paper documents.

2.3 Document Understanding

The reverse process of document production is called *document understanding*. It is applicable on every kind of printable documents such as captured images (scanned documents) or electronic documents (PDF documents, synthetic images). Document understanding tries to recover the original meaning from printed documents. The process leading to document

understanding is carried out in multiple steps, such as image preprocessing, physical structure recognition, and logical structure recognition (see Subsection 2.3.1, 2.3.2, and 2.3.3 respectively). Figure 2.6 shows a schematic diagram of the document understanding process. Let us note that physical structure recognition and logical structure recognition may sometimes be mixed in a unique recognition process. Actually, document understanding can be seen as a reverse-engineering task using a fixed-layout document as input.

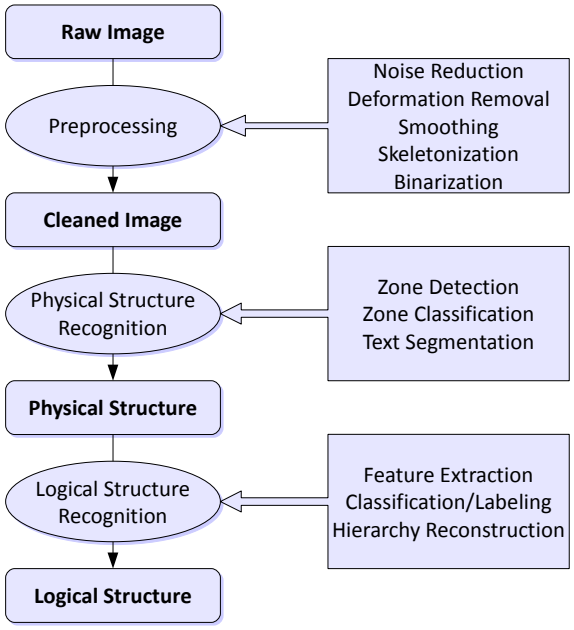


Figure 2.6: The common document understanding process is divided in three main steps: image preprocessing, physical structure recognition, and logical structure recognition.

2.3.1 Image Preprocessing

Starting from a physical printed document, the first step is to digitize the hard copy in order to get an electronic version of the document, i.e., a raw raster image. Such acquired document images contain noises and artifacts that are mainly introduced during the document digitizing or scanning phase and may be caused by dust, poor scanning conditions, or paper deterioration.

Image deformation is also common when digitizing paper documents, it is mainly due to the way the paper is handled and processed by the scanner. Thus, in order to make the analysis algorithms more robust to noise and deformations, a first *image preprocessing* phase is applied on the digitized images, that tries to correct images imperfections and prepare them for future high-level processes. The preprocessing phase may contain many steps such as noise reduction, skew detection and correction, foreground/background separation, image

smoothing, binarization, and skeletonization.

2.3.2 Physical Structure Recognition

The *physical structure recognition* aims at decomposing a printed document into a hierarchy of homogenous regions, such as images, graphics, and text blocks. Methods for the physical structure analysis fall roughly into three categories, depending on their approach.

- *Bottom-up* algorithms start with the smallest components of a document (pixels or connected components) and repeatedly group them to form larger and homogeneous regions, e.g., characters are merged into words, themselves merged into lines, themselves merged into blocks. The key advantage of bottom-up methods is that they can handle arbitrarily shaped regions with ease. But the drawback is that they struggle to take into account higher-level structures in the image, such as columns. This often leads to overfragmented regions.

- *Top-down* algorithms start with the complete document image and divide it recursively into smaller components until satisfying homogeneous regions have been detected (given a predefined homogeneous criterion). Although top-down methods have the advantage that they start by looking at the largest structures on the page, they are unable to handle the variety of formats that occur in documents having complex layouts, such as magazines or newspapers.

- *Hybrid* algorithms combine the two approaches mentionned above, i.e., bottom-up and top-down, in order to improve the segmentation by using a split-and-merge strategy.

Many different algorithms have been developped using bottom-up, top-down, and hybrid approaches; they are presented in details in Chapter 3 “State of the Art”.

2.3.3 Logical Structure Recognition

The *logical structure recognition* is the reverse process of document formatting. This phase tries to recover the logical structure of a document from its physical structure. Thus, the restructuring process is highly dependent on the formatting rules used during the document authoring. For instance, titles are often rendered with big font sizes whereas paragraphs use small font sizes and well defined layouts.

The logical structure recognition aims at recovering the set of logical entities in a document, along with their inter-relationships. It is performed on the results of the physical structure recognition and generally proceeds in two successive phases:

- The *labeling phase* aims at assigning logical labels to the physical objects recovered during the physical structure recognition process.

- The *hierarchy reconstruction* aims at recovering the hierarchical structure of the logical objects, in order to reconstitute the entire logical structure of the document.

The logical structure recognition also focus on the notion of reading order, which is a sequencing of the textual contents that makes comprehension of the document easier. Langagues such as Arabic or Chinese can have different reading directions (right-to-left, top-to-bottom) than Latin languages. Thus, multilingual restructuring systems have to take such parameters into account. Chapter 3 "State of the Art" presents in details the existing algorithms used to recover the logical structure of documents.

2.4 Document Models

Each document has one or more inherent logical structures that can be mapped to a large variety of different layouts by means of formatting processes and by using appropriate styling rules. At opposite, a unique physical structure formalism is needed to describe all the possible document formatting. Although a formatted document certainly reveals its underlying logical structure, the back conversion from the physical to the logical structure is not easily feasible, because a formatting process is not reversible in a mathematical sense. A *document model* is precisely a knowledge base used to recover the logical structure of a document from the physical one.

In other words, a document model can be seen as a set of transformations describing the structural constraints and relations of a specific class of document in order to do the backconversion from the physical structure to the logical structure.

Figure 2.7 shows a schematic representation of a document model: a knowledge base is used to classify and link together the physical objects in order to recover the underlying logical structure.

The major issue is the following: "How can such models be generated?". Various schemes can be used in order to produce document models; their generation may be implicit, explicit, or inferred from examples:

- *Implicit document models* are directly hard-coded in the application, and thus are difficult to modify, adapt, or extend. This technique may be useful for very specific applications focusing on a particular class of documents, e.g., address detection and recognition on postal letters.

- *Explicit document models* are written by an expert thanks to a predefined (and thus restrictive) document model language, they enable the creation of new models by a human operator. Nevertheless, producing a new model is a complicated task and requires a high degree of expertise.

- *Inferred document models* are generated from examples, they are flexible but need the

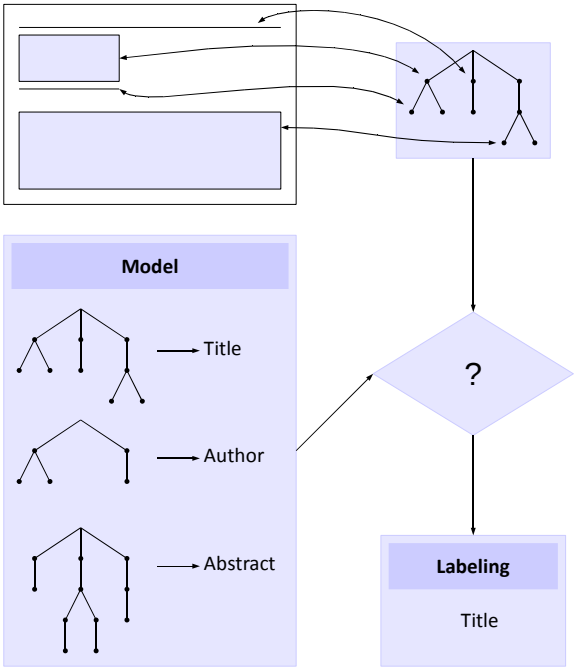


Figure 2.7: A document model is basically a knowledge base used to reconstruct the underlying logical structure of a document. This figure shows a schematic representation of a document model where a given pattern is recognized thanks to a similarity measure relative to a set of ground-truthed data.

development of a complete learning system relying on representative ground-truthed training data.

Regarding the document model knowledge, it may be represented under various forms: statistical models, stochastic models, and structural models.

- *Statistical models* use statistical information extracted from training samples. Various learning and classification methods are based on statistical features such as naive Bayesian classifier, k-nearest-neighbor, parzen windows, neural networks, or support vector machines.

- *Stochastic models* consider the sample to recognize as a sequence of states. The model describes the states thanks to transition probabilities and observation probabilities. The classification is done by finding the path that maximizes the current event probability, e.g., hidden Markov models using a Viterbi path algorithm. However, stochastic models impose documents to be linearized in order to be described as sequences of states.

- *Structural or syntaxic models* allow structural and contextual data to be described thanks to a graph structure or a grammar. Classification then consists in comparing graphs or verifying if a given sequence is an existing word in the grammar.

Document model generation is a very difficult task, that can be achieved using many different techniques. Thus, in Chapter 7, we present Dolores, a logical restructuring environment we have developped, which precisely produces document models thanks to an elaborated and interactive user-interface. First of all, in the next chapter, i.e., Chapter 3, we present the state of the art about document analysis methods.

3

State of the Art

Contents

This chapter summarizes the state of the art tools and researches in the field of document analysis. The first section, Section 3.1, gives a brief overview of the document image analysis history. Then, Section 3.2 and Section 3.3 offer a look of the existing techniques used to recognize the physical and the logical structures of digitized documents, respectively.

Afterwards, in Section 3.4, we present some complete document image analysis systems. Section 3.5 focuses on restructuring systems using PDL documents as input as they are directly related to our research work. Finally, a review of the existing document recognition data formats is exposed in Section 3.6.

3.1 A Historical Overview

Researches about document image analysis have first been investigated at the beginning of the twentieth century. The beginnings of *optical character recognition* - OCR - can be traced back to 1809 when the first patents for devices to assist visually impaired people were awarded. An OCR-like machine appeared in 1914 when Fournier D'Albe invented a device called the optophone [68]. The optophone produced sounds corresponding to specific letters or characters. Thus, after learning the character equivalent for various sounds, visually impaired people were able to read printed material.

The first real OCR machine was actually invented in 1929 by Gustav Tauschek; it was a simple mechanical device that used character templates. A photodetector was placed in such a way that when the template and the character to be recognized were lined up for an exact match with a light directed towards them, no light would reach the photodetector.

Since the dawn of the modern computer era, in the 1940s, more sophisticated document image analysis systems appeared successively. Among the first systems that were to be developped arose specific recognition tasks such as automatic mail processing (address recognition) or postal check recognition. During the last decades, document image analysis systems were traditionally dealing with scanned images of printed material in order to extract their content and structures. A detailed review of such existing document analysis algorithms is presented in the paper "Document Structure Analysis Algorithms" [70] by Mao et al. However, systems using directly the electronic document content as input, e.g., description language formats such as PDF, appeared only recently.

Nowadays, and according to the document engineering community, recognition systems usually distinguish two levels of document structures. The lower level is called the physical structure and refers to the layout and typographic entities composing a document. At this level, no information is provided about the interpretation of such entities. This aspect is revealed by a second abstraction level that is called the logical structure. The two next sections precisely focus on the recognition of such document structures.

3.2 Physical Structure Recognition Methods

As presented in Subsection 2.3.2, the physical structure recognition methodologies are grouped in three main approaches: bottom-up, top-down, and hybrid. The results of the physical structure recognition are then generally represented as a set of labelled regions or zones such as text blocks, images, and graphics.

3.2.1 Bottom-up Approaches

Bottom-up approaches proceed by merging low-level document primitives, i.e., pixels or connected components, in order to generate low-level document structures. A possibility consists in the extraction of features from the primitive components in order to group them into homogenenous regions. Such methods are presented in several key papers [34, 36, 41, 62].

Another existing possibility is to process the original image through some region filters. In the paper entitled "The Document Spectrum for Page Layout Analysis" [79], O'Gorman presents Docstrum: a system that does not use the connected components homogeneity as criterion any more, but instead clusterizes them by analyzing their neighborhood relationships (distance, angle). Techniques such as RLSA [120] and voronoi diagrams [58] have also proven to be successful in order to group together primitives into regions.

3.2.2 Top-down Approaches

Top-down approaches start from the entire document image and try to decompose it iteratively into smaller regions. Gatos [39] and Antonacopoulos [7] focus on page segmentation using image background: white connected components are detected and then segmented using a contour following algorithm. Breuel's [16] algorithm for layout analysis proceeds in two key steps: finding a cover of the background whitespace of a document in terms of maximal empty rectangles, and finding constrained maximum likelihood matches of geometric text line models in the presence of geometric obstacles. As Gatos and Antonacopoulos, this algorithm also uses page background, but it is considerably easier to implement than prior methods, returns globally optimal solutions, and requires no heuristics. Krishnamoorthy [61] and Wang [112] use the X-Y Cut algorithm: the image is decomposed recursively by using projection profiles. Baird et al. [10] also developed an original image segmentation method using shape-directed covers able to analyse complex printed-page layouts. Methods using image multiresolution have also been investigated, e.g., Cinque [20] proposes to extract features (mean, standard-deviation) on subwindows generated from various image downsamples in order to recover the image background.

3.2.3 Hybrid Approaches

Hybrid approaches try to combine the best of bottom-up and top-down methods. Pavlidis and Zhou [83] advanced a hybrid algorithm using a split-and-merge strategy: first, a top-down approach is used to come to an over-segmented document image, then, regions that are similar and nearby are merged to form homogeneous regions.

In his PhD thesis, Azokly [8] presents a system combining both an algorithm based on the hierachical cut of structuring rectangles with a rule-based algorithm merging components given predefined structures. Differently, Smith [99] first uses a bottom-up phase via tab-stop detection to deduce the column layout of a page and then applies a top-down phase to impose the final structure and reading-order on the previously detected regions.

3.3 Logical Structure Recognition Methods

As presented in Subsection 2.3.3, the logical structure recognition usually proceeds in two key phases: physical blocks labeling and hierarchical tree structure reconstruction. The labeling phase aims at tagging physical blocks with logical labels that define the function of such blocks in the document. These labeled blocks are then reorganized in a hierarchical tree structure, keeping track of the reading order. The labeling phase proceeds itself in two phases: features extraction and classification.

Thus, logical structure recognition systems described in litterature differ in a) the extracted features, b) the classification method used, and c) the hierarchical structure reconstruction algorithm.

3.3.1 Feature Extraction

Lots of features may be extracted from document images. Features that directly relates to segmented blocks can be divided in two categories. Some features may be relative to the intrinsic properties of a block (self-related attributes), other may use neighborhood relations, or position on the page (cross-related attributes).

Self-related Attributes

Self-related attributes define only the intrinsic features of a block. These might consist of *morphological features* such as the block dimension, elongation (width/height ratio), density (black/white pixels ratio), etc. Wang and Srihari [112] precisely used morphological features such as black and white run-length transitions to classify their block.

Structural features describe textual blocks attributes such as number of lines, interlines, and typographical attributes such as justification (left, right, centered, justified), interline, font size, font name, font face (bold, italic, underlined).

Textual features consist in defining a set of words or in using regular expressions in order to detect precise entities (numbers, addresses). For example, Ishitani [56] detects list structures thanks to word patterns such as "2.", "(1)", "a)", or formula thanks to mathematical symbols. Klink [60] uses sequences of words and regular expressions to recognize and extract specific blocks of text.

Rangoni [95] combines morphological, structural, and textual features as input features for an artificial neural network used to recognize the document logical structure.

Cross-related attributes

Cross-related attributes express features extracted from relation between blocks. The *geometrical relations* between blocks are often used as features, e.g., it is possible to express "block A must be underneath B". Geometrical regions can either be qualified relative to the observed object or through document coordinates. Allen's [4] relative positions are defined using a two-dimensional time-relation. Azokly [8] goes further and defines 169 types of relative positions between two blocks (see Figure 3.1).

The *textual relations* specify the absolute or relative number of common words between blocks. A list of potential common words that need to be located in the related blocks may also be predefined. The *label relations* enable a cross related block to be qualified through other specific labels. Label conditions may be very useful, particularly in regard to document objects that are depending on each other. Klink [60] uses such label relations in order to induce cross-related block labels, i.e., the occurrence of one logical block triggers the labeling of other blocks. For example, if the "title" is found on a scientific paper, then the "author" (or an "abstract") is usually found underneath the "title". Thus, if no "title" exists on the document, no "author" (or "abstract") will exist either; eventually, the precision of the

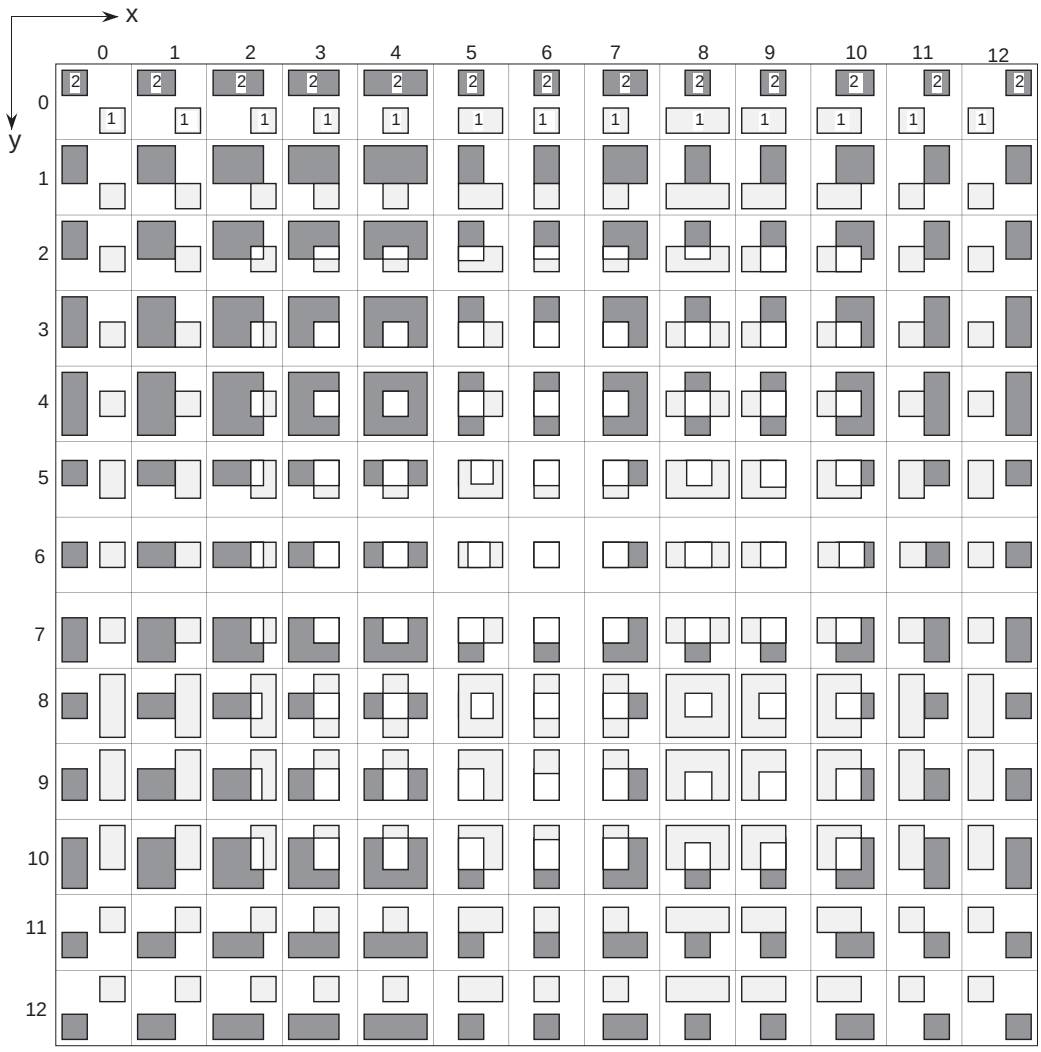


Figure 3.1: The above figure shows the local topologies defined by Azokly: 169 types of relative positions between two blocks are presented.

labeling will increase. Note that label relations imply an heuristic approach since the logical label is precisely the researched information.

Cinque [20], Spitz [100], and Belaïd[12] simply use recurrent absolute positions between blocks in order to recognize common blocks such as headers or footers.

3.3.2 Classification

The purpose of the *classification* phase is to assign labels to the previously segmented blocks. The classification task is done by a *classifier*: a system that uses the features extracted from a block in order to classify it in a predefined class. A classifier needs a certain amount of knowledge about the analysed document in order to labelize (classify) each block. This knowledge is directly related to the formatting rules applied during the production process.

Depending on the class of documents (and its complexity), the extracted features, and the purpose of the recognition system, the classification methods may be quite different. Many classification methods simply encapsulate the knowledge thanks to hard-wired rules. For example, Cinque [20] uses rules such as "a heading has to be composed of at most two rows and must not exceed one quarter of the vertical dimension of the page" to labelize text blocks.

More elaborated classifiers are trained on ground-truthed data in order to automatically infer their knowledge. Indeed, features extracted from a block are commonly represented as vectors in a multidimensional space. The classification then consists in partitioning the feature space into classes corresponding to predefined block labels. Such classifiers may use basic methods like the Bayesian rules or the k-nearest-neighbor paradigm. For instance, Héroux [47] computes the mean vector of morphological features for each class. The classification then simply consists in matching each block vector with the nearest class mean vector.

Esposito [33] classifies the segmented blocks thanks to artificial neural networks (ANN); thus, the classifier knowledge is contained in the topology and the neuron weights. Rangoni [95] goes a step further by improving the artificial neural network topology with intermediate transparent layers and by applying iterative perceptive cycles.

Robadey [97] uses homogeneity and discrimination criteria to infer a decision tree based on a predefined set of features. The construction of the decision tree is achieved thanks to iterative specialization and extension steps during an interactive learning phase: the user first corrects a classification error, then the system analyses the correction and adapts its model, i.e., its decision tree, as a consequence.

3.3.3 Hierarchical Structure Reconstruction

Tsujimoto [109] hard-coded only four rules allowing any physical tree structure to be transformed into a logical tree structure. His system is very generic, but works only on documents having standard and non-complex layouts.

Graph structures and comparison have been investigated by Walischewski [110] and Héroux [47]. Héroux infers a model by computing the isomorph subgraph for each class of learning samples. The classification of a block is done by computing its distance with each graph of the model (corresponding to each class). On the opposite, Walischewski describes all the document with a single graph. The document model is a synthesis of all the learning sample graphs: it groups together the nodes of each graph and assigns them a probability. Classification then consists in finding the most likely isomorphic subgraph between the model and the samples (given the node probability).

Belaïd [12] represents the hierarchical logical structure through a grammar that is automatically inferred from a set of labeled physical blocks (training set). Hu [51] also proposes a document model described by a grammar where the logical structure of a document is expressed via production rules. In the article entitled "Modelling the Logical Structure of Books and Journals Using Augmented Transition Network Grammars", Stehno [101] attempts to describe the rules governing the layout of books and journals in terms of syntactical and abstract layout relations by using augmented transition network grammars.

Some systems provide methods allowing previous results to be reconsidered. Brugger [17] and Palmero [82] use systems based on statistical data which are able to learn from samples. Their strategy allows the physical blocks labeling to be fuzzy and thereby reconsidered by the logical structure reconstruction algorithm. Thus, transformation of a physical tree into a logical one is done by choosing the best solution between a set of possible ones.

In his approach, Brugger represents the logical tree structure of documents by probabilities of local tree node patterns similar to n-grams. Since n-grams are usually used to represent sequence structures, e.g., in natural language processing or genetic sequence analysis, Brugger generalized their mechanism in order to apply them to tree structures. Thus, during the structure recognition, several alternative tree paths are kept in memory: the one that best optimizes a predefined criterion is first explored. However, in case of failure, an alternative solution is researched.

Ishitani [56] and Niyogi [77] divide their systems into modules, each of them encapsulating a single recognition step. The logical structure recognition is then achieved by a core module that handles the interaction between the recognition modules and that allows unsatisfying results to be reconsidered. In another recognition system, Yamaoka [125] and Niyogi [77] merge the labelization and hierarchical reconstruction phases. As follows, labeling and structure reconstruction proceed simultaneously and together, such system benefits of the knowledge gained in each iterative phase.

LeBourgeois [64] tries to reconstruct the hierarchical structure by first inferring logical labels with confidence rates to the segmented blocks. Then, the logical hierarchy recognition system uses an heuristic called *probabilistic relaxation*: an iterative algorithm that tries to optimize a global solution thanks to local measures. This method was first described by Rosenfeld [98] and allows the classification to be iteratively reconsidered and improved in

order to propose a set of local solutions matching a global description.

3.4 Document Image Analysis Systems

Until now, we only presented single techniques used for DIA - document image analysis -, i.e., image preprocessing, physical structure recognition, and logical structure recognition. On the opposite, *document image analysis systems* group together some of these techniques in order to provide complete DIA workflows. Such systems have recently appeared [5, 66, 3, 124], they usually focus on well defined categories of documents.

For instance, Aiello et al. [3] elaborated a basic document analysis system which is able to assign logical labels and extract the reading order in documents having standard layouts. All information sources, from geometric features and spatial relations to the textual features and content, are used in the analysis. To deal effectively with these information sources, they define a document representation general and flexible enough to represent a "broad set" of documents. Their recognition method uses generic document knowledge in order to identify explicitly a predefined set of label classes. That is, their system is based on many hypothesis, such as "title font sizes are bigger than paragraphs" or "page numbers are at the end of a page", and thus is applicable on documents having common layouts.

Altamura et al. [5] developed WISDOM++, a document processing system that can transform paper documents into XML-based descriptions. WISDOM++ is a knowledge-based system capable of supplying assistance to the user during the document analysis and recognition process. Its knowledge base is declaratively expressed in the form of decision trees or rules and is automatically built from a set of training documents using machine learning techniques.

More specific works have also been successfully done. For example, in the paper entitled "RASADE: Automatic Recognition of Structured Document Using Typography and Spatial Inference" [65], Lebourgeois and Souafi-Bensafi describe RASADE; an industrial project aiming at recovering the logical layout of content tables in order to automatically index books and magazines. Yacoub et al. [124] describe a complete system for the analysis of a large collection of complex documents, i.e., magazines and journals. Their system also tries to reconstruct the logical structure based on the recognition of the table of contents. In the paper entitled "A Prototype Document Image Analysis System for Technical Journals" [75], Nagy et al. describe a DIA system offering a complete journal browser, which is specifically designed for technical journals.

3.5 Electronic Document Analysis

Today, almost 90% of published or exchanged electronic documents uses the PDF format. Consequently, the need for reusing their content has incited the development of tools able

to access, extract, and convert their data. Subsection 3.5.1 precisely presents some existing PDF extraction and analysis tools. Then, researches about the recognition of PDF structures are presented in Subsection 3.5.2.

3.5.1 PDF Extraction and Analysis Tools

Several tools have been designed and developed to manipulate PDF files. For instance, Adobe provides PDF producers, readers, and even an SDK - software development kit - used by developers to create softwares and plug-ins able to interact with PDF documents or customize existing Adobe products. Furthermore, third party developers propose various PDF softwares and APIs - application programming interfaces -, either under license or as freeware. Consequently, a wide range of PDF tools are proposed in the market. Table 3.1 presents a possible classification for these tools (see Planet PDF [85]).

Category	Description
Creation & Conversion	Create and convert to PDF files from various file types.
Developer	Developer libraries for accessing, editing, and creating PDF files.
Editing & Management	Manage, edit, update, modify, and manipulate.
Extraction	Extract text, images, and data from PDFs to other file formats.
Forms & FDF	Create, edit, collect, and extract data from forms.
Imposition & Color	Check, separate, and correct colors. Order and re-order pages.
Links & Bookmarks	Add, edit, and manage navigational elements.
Prepress & Print	Prepress, preflighting, color management, trapping, and imposition.
Searching & Indexing	Meta data, document info, indexing, and searching.
Security & Copyright	Secure and protect PDF files. Restricted use and access control.
Server-side	Server-based software for conversion, manipulation, and delivery.
Split & Merge	Split, merge, append, and collate.
Stamp & Watermark	Add stamps, watermarks, and impressions.
Viewers	Open, view, access, and edit PDF files.

Table 3.1: Categorization used by Planet PDF in order to browse their PDF-related products database. We clearly observe that a wide range of PDF tools exist on the market.

Planet PDF's database indexes more than 600 products, each of them might be classified in more than one category at the same time. In this large amount of PDF products, our concern mainly focuses on the extraction tools, able to extract and interpret either the entire content from PDF files, i.e., text, graphics, and images, or a well-defined subset of their primitives. Most of the extraction tools are able to convert their results into various formats, easier to manipulate than PDF. Such formats provide edition functionalities and allow the document's content to be reused, to the best of their extraction.

Able2Extract [1], PDF Transformer [107], PDF2Office [88], and PDF-File Converter [22] are precisely extraction softwares allowing one to convert the content of PDF documents into Microsoft's Word format. This solution has certainly been chosen in order to enable the

reedition of converted documents with a widespread application. Furthermore, such output format allows extraction tools to reproduce as accurately as possible the original layout of documents.

Tools such as Able2Extract, PDF Transformer, and PDF2Office also convert PDF files into HTML, whereas PDFTron [90] is specialized for producing SVG documents (HTML and SVG are presented in Subsection 3.6.3). Both formats have been developed for publishing on the Internet and allow documents with complex layouts to be reproduced in a reliable way. Moreover, HTML and SVG are handily formats (XML-based), easy to read and manipulate for both users and computer applications.

Mimotek [102] goes a step further by making a deeper analysis of PDF pages to determine their underlying logical structures. However, this tool has to be manually configured and customized by the Mimotek engineers in order to deal with new document classes. High-level extraction facilities are also provided by xtPDF Extractor [123]. This tool allows the PDF content to be extracted and restructured given a predefined and generic logical structure. This predefined logical structure may be customized manually thanks to a basic scripting tool, by specifying rules based on style, location, and content.

Finally, applications such as PDFText [89] and TEXTfromPDF [105] only extract the PDF textual content into ASCII codes. The output document does not keep any information about the other primitives, i.e., graphics and images, as well as the original layout. Such applications can however be useful for indexing systems and applications working with raw text streams.

Table 3.2 shows the results of a comparison we did over a selection of some of the most used PDF extractors and converters. The rating of the PDF tools has been made thanks to four criteria, going from low- to high-level. At the lowest level stands *Textual* primitives extraction, meaning that a tool is able to extract text primitives correctly and store them in a raw text file. *Graphical* primitives extraction includes text, graphics, and images extraction. Such PDF products are able to read various image formats, deal with vector graphics and font programs in order to convert the extracted primitives into new document formats (SVG, HTML, etc.). The *Physical* structure criterion indicates that a phase of document analysis is performed to extract information about the document's layout. Finally, at the highest level stands the *Logical* structure criterion, meaning that a tool is designed to recover the logical structure of predefined classes of documents, e.g., magazines, newspapers.

Boundary lines are sometimes hard to determine, especially in the case of physical structure extraction. Some tools pretend to extract this structure, whereas this extraction performs efficiently only on documents having simple layouts. Moreover, the chosen criteria do not reflect many integrated particular features. For example, few PDF extractors are able to automatically remove inconsistencies from PDF files, such as non-sense blanks or odd text segmentation. PDF extractor tools may also offer interactive windowed environment with WYSIWYG editors or even provide application programming interfaces allowing soft-

URL and Reference	PDF Tool Name	Textual	Graphical	Physical	Logical
www.processtext.com [21]	ABC Amber PDF Converter	•			
www.investintech.com [1]	Able2Extract	•	•	•	
www.adobe.com [54]	Acrobat Standard	•	•	•	
www.docsmartz.com [92]	DocSmartz Pro	•	•		
www.pdfdesk.com [26]	Easy PDF Creator	•	•		
www.getpdf.com [40]	GetPDF	•	•		
www.iceni.com [32]	INfix PDF Editor	•	•	•	
www.jpedal.org [57]	JPedal	•	•		
www.mimotek.com [102]	Mimotek Structuriser	•	•	•	•
www.nitropdf.com [84]	Nitro PDF	•	•		
www.recosoft.com [88]	PDF2Office	•	•	•	
www.cadkas.de [121]	PDF 2 Word	•	•		
www.pdf-convert.com [87]	PDF-Convert	•			
www.pdfkit.com [59]	PDF Export Kit	•	•		
www.pdf-file.com [22]	PDF-File Converter	•	•	•	
www.pixelplanet.com [42]	PDF Grabber	•	•		
www.qoppa.com [103]	PDF Studio	•	•		
www.pdf-analyzer.com [89]	PDFText	•			
www.filehunter.com [24]	PDFText Converter	•			
www.pdftransformer.com [89]	PDF Transformer	•	•	•	
www.pdftron.com [90]	PDFTron	•	•		
www.pdf2text.com [23]	PDF XML Converter	•			
www.archisoftint.com [86]	Solidconverter PDF	•	•		
www.pdftodocconverterpro.com [25]	Smart PDF Converter	•	•		
www.textfrompdf.com [105]	TEXTfromPDF	•			
www.mesadynamics.com [108]	Trapeze	•	•	•	
www.verypdf.com [106]	VeryPDF Tools	•	•		
www.glyphandcog.com [122]	Xpdf	•	•		
www.exti.com [123]	xtPDF Extractor	•	•	•	•

Table 3.2: Comparison of existing PDF tools. The "•" symbol indicates that a tool efficiently integrates the functionality. Textual, Graphical, Physical, and Logical stand for textual primitives extraction, graphical primitives extraction, physical structure recognition, and logical structure recognition, respectively.

ware engineers to create new applications accessing PDF files' content through the extractor interface.

3.5.2 PDF Analysis Researches

The document analysis field has also been stimulated by the success of PDF: in the recent past, various fundamental researches have been accomplished in order to recover the original layout and logical structures of electronic documents. Related researches belong to two main categories, the former analyzing raster images and the latter taking advantage of the extracted PDF primitives.

Raster Image Analysis

By rendering graphical primitives of a PDF document on a high-resolution raster context, high quality synthetic bitmap images, not affected by noise and skews, can be generated. Standard techniques of document image analysis and recognition can be successfully applied to synthetic images. In that sense, Doermann [29] was an avant-gardist: even before the advent of the PDF file format, he applied image analysis methods on DVI - device independent file format - files rendered as bitmap images, while profiting of precious information hold in the DVI description.

A key advantage of image analysis methods consists in being independent from the internal structure of PDF files: they can be applied to any kind of documents, either containing scanned pages (thus losing the interesting synthetic property) or composed of electronic primitives. But, above all, these methods benefit of the experience gained in the field of document image analysis during the last decades. Synthesis and reviews of such document analysis methods have already been presented in Sections 3.2 and 3.3, and described in several major publications [120, 80, 30, 74].

Electronic Content Analysis

The second category of works takes benefit of the electronic primitives composing the document. It has to be said that quite few scientific papers have been published so far about PDF physical and logical structure recognition.

In 1998, Paknad and Ayers [81] deposed a U.S. patent for a basic PDF word segmentation and word listing technique that can be used to search for words in PDF files. The inferred structures were transient, i.e., they were only present in memory during the processing time and thus lost once the searching done. However, since PDF version 1.4, physical and logical structures can be integrated in the so-called "tagged PDF" thanks to a dedicated and internal PDF structure tree. Therefore, various works used this straightforward scheme to directly embed the recovered structures in the original electronic document [9, 45, 69]. Such a method

has the advantage of preserving the information contained in the source PDF documents while adding the recovered structures.

For instance, Bagley et al. [9] define a PDF structure called COG - Component Object Graphics -, a structure which encapsulates a set of graphic data belonging to a unique graphic entity. However, they do not provide a restructuring mechanism since these COG elements are directly extracted from the authoring software by a pluggin added to the PDF writer. In the paper entitled "Creating Structured PDF Files Using XML Templates", Hardy et al. [45] describe a tool for recombining the logical structure from an XML document with the typeset appearance of the corresponding PDF document. That is, an XML version containing both the text content and the logical structure must first be supplied. Then, a basic process is used on the one hand to break the content of the PDF document down into individual characters (a necessary step because there is no guarantee that explicit word boundary demarcation will occur) and, on the other hand, to group these characters into lines in reading order. A match is hence achieved between the XML and PDF contents, allowing the logical structure to be injected in the original PDF file. The XML content is also used to correct word boundary demarcations and add white spaces in the PDF document.

In their paper entitled "Document analysis of PDF files: methods, results and implications", Lovegrove and Brailsford [69] introduce a strategy for document analysis which uses PDF as its starting point. Their work is probably the most advanced one concerning the PDF physical structure recognition. Indeed, they developed a basic algorithm able to reconstruct words, lines, and paragraphs, based on the font size, font name, and bounding boxes. A generic logical structure has been defined by using the following predefined block labels: graphics, main text, captions, images, titles, headers, and footers. The recognition is handled by a set of restructuring rules and criteria that have first been inferred during a training phase examining the recurring properties of each block label.

A different solution is proposed in various works [6, 18, 19, 28, 38, 94]: instead of storing the analysis results in the original file, the inferred structures are expressed in an XML-based format. This solution is well suited for further analysis or for reusing documents, because both human beings and automatic systems can easily manipulate XML data.

Anjewierden and Kabel [6] suggest to extract logical structures from PDF files by detecting layout components and by classifying them using a set of rules. Logical structures are then associated to ontologies, in order to index the documents' content. Déjean and Meunier [28] try to recover the logical structure of PDF documents by first recognizing their table of contents and by using it to reconstruct their logical hierarchy. Futrelle et al. [38] come up with a technique for classifying bar-graphs and non-bar-graphs diagrams, extracted from PDF files, by analyzing the graphical primitives composing the documents. Classification is achieved thanks to a binary SVM - Support Vector Machine - processing statistics (forming feature vectors) extracted from PDF graphics.

The works of Chao aim at recovering the physical structures of PDF documents, in or-

der to separate the information in the foreground from that in the background [18], and to adapt the document's layout to the rendering constraints or printing device specificities [19]. Rahman [94] proposes a tool for recovering the physical structures of PDF files using geometrical properties. That is, the layout analysis is based on the idea of separating content of documents based on the detection of white space and black space rectangles and by recursively merging them. Each detected block is then classified into a distinct category, i.e., text, images, lists, and tables. The processed PDF files and the layout analysis results are finally converted into HTML documents.

The techniques developed for analyzing electronic content may give very good results because they can take full benefit of the exact description of primitives. Unfortunately, they are completely inefficient in the case of PDF files containing scanned pages. Consequently, in previous works [43, 14], we proposed to mix electronic content analysis with image analysis, in order to overcome this problem.

To summarize, all the works presented in this section target at extracting physical or logical information from PDF files. Major part of the systems presented above store the extracted structures into XML descriptions. Most of these XML descriptions only store the documents structures, some of them also do include their content, but none of them provides a solution for their layout. Thus, XML files often contain a reference to the original PDF documents, in order to keep the original information. Physical structures and PDF contents are sometimes stored as HTML files, thereby also loosing the exact document layout. Each of these works focuses on a specific task; henceforth, none of them does provide a general solution.

3.6 Document Recognition Data Formats

Many document formats exist, but few of them allow content, physical stucture, and logical structure to be reliably expressed. For instance, page description languages such as PDF [53], DVI [113], PostScript [2], or XPS [119] have not been specifically designed to keep or express physical and logical structures. On the opposite, DAFS [93], developped by RAF Technology, is a complete file format designed uniquely to store the physical and logical structures resulting from document recognition softwares. Other file formats have also been developped to store complete document structures such as METS/ALTO [13] and ODA [50] whereas XDOC [52] and DjVu [44] are only able to represent scanned document results. Some file formats such as DocBook [111] and LaTeX [63] clearly separates the content and logical structure from the layout, thus providing pure authoring tools.

This section is divided into three subsections. First, Subsection 3.6.1 presents two generic file formats, SGML [91] and XML [46], which are two meta-languages used to represent any kind of data. Then, Subsection 3.6.2 exposes some file formats used to represent document image analysis results. Finally, Subsection 3.6.3 emphasizes some of the existing standard

document file formats.

3.6.1 Generic File Formats

SGML-Based Formats

The Standard Generalized Markup Language [91] - SGML - is an ISO-standard technology used to define generalized markup languages for documents. The basic design goal of SGML is to provide an opened formalism which ensures that documents encoded according to its provisions should be transportable from one hardware and software environment to another without loss of information. Thus, SGML has been regularly used to store document recognition results: one of this attempt is ODIL [67] which is used in the PRASAD project. Nowadays, SGML has mainly been replaced by XML-based format, a revised subset of the SGML language.

XML-Based Format

The Extensible Markup Language [46] - XML - is a markup language for creating special-purpose markup languages (thus XML is a meta-language), capable of describing any kinds of data. It is a simplified subset of SGML: its primary objective is to facilitate the sharing of both the format and data across different systems, in particular systems connected via the internet. XML's design purposes emphasize simplicity, generality, and usability over the Internet.

There is a variety of programming interfaces which software developers may use to access XML data, as well as several schema systems designed to assist in the definition of XML-based languages. According to Wikipedia [117]: "As of 2009, hundreds of XML-based languages have been developed, including RSS, Atom, SOAP, and XHTML. XML has become the default file format for most office-productivity tools, including Microsoft Office, OpenOffice.org, AbiWord, and Apple's iWork."

Indeed, XML has many strenghts: it is a platform-independent format, it is a human- and machine-readable format, it is self-descriptive, its syntax is simple and strict, and, last but not least, it is a standard file format. However, XML suffers also from one major weakness: its syntax is fairly verbose and partially redundant. Thus, XML is rarely used to represent heavy file formats such as image, music, or movie formats.

3.6.2 DIA File Formats

XDOC

XDOC [52] has been specifically designed to capture the content and structure of scanned documents containing both text and graphics. In addition to recognized text, XDOC provides detailed information about page layout, i.e., x- and y-coordinates of text and graphics on

the page, character formatting (bold, italic, subscript, superscript, underline), font metrics, images, horizontal and vertical rulings, document structuring conventions, lexical classes, character and word confidence, word bounding boxes. XDOC is an old file format developped to store the content of paper documents as recognized and output by ScanSoft Inc. document recognition software. Thus, it is tight to OCR output results and therefore lacks a generic document representation. Moreover, the internal file description is non-standard, it uses ASCII constructions in order to represent content and structures.

DjVu

As XDOC, DjVu [44] is a computer file format designed primarily to store scanned documents, especially those containing a combination of text, line drawings, and photographs. It uses technologies such as image layer separation of text and background/images, progressive loading, arithmetic coding, and lossy compression for bitonal (monochrome) images. DjVu provides techniques to store high quality and readable images in a minimum of space, so that they can be made available on the web.

DAFS

A widely spread representation format for document recognition results is the Document Attribute Format Specification [93] - DAFS - which is influenced by standards such as SGML and ODA. It has been designed for document and image understanding and, thus, provides mechanisms to represent the physical as well as the logical structure of documents. A DAFS document is a hierarchical structure, whose nodes are called entities. Each entity represents a document component such as a block, a line, or a word. An entity has a content and arbitrary key-value pairs called properties; an entity references an image on which it occupies a polygonal area. Unlike SGML, DAFS is a binary file format, it is compact, and capable of storing image and data in one file. The current DAFS implementation is no longer officialy supported, so it becomes quite a questionable decision to make a new system rely on it.

METS/ALTO

ALTO [13] - Analyzed Layout and Text Object - is an extension schema to METS - Metadata Encoding and Transmission Standard -. It is a standardized XML format that describes the layout and content of documents. METS provides metadata and structural information while ALTO contains only the content and physical information. ALTO is specifically used to store layout information and OCR recognized text extracted from pages of any kind of printed documents like books, journals, and newspapers.

3.6.3 Document File Formats

LaTeX

LaTeX [63] is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents but it can be used for almost any form of publishing. LaTeXis based on the idea that authors should be able to focus on the content of what they are writing without being distracted by its visual presentation. In preparing a LaTeX document, the author specifies the logical structure using familiar concepts such as chapter, section, table, figure, etc., and lets the LaTeX system worry about the presentation of these structures. Therefore, it encourages the separation of layout from content while still allowing manual typesetting adjustments where needed.

DocBook

DocBook [111] is an XML semantic markup language for technical documentation. It was originally intended for writing technical documents related to computer hardware and software, but it can be used for any other kind of documentation. As a semantic language, DocBook enables its users to create document content in a presentation-neutral form that captures the logical structure of the content. That content can then be published in a variety of formats, including HTML and PDF, without requiring users to alter the source. DocBook focuses on the logical structure of a document and, thus, does not address the layout and typographic rules. Document layout is left to stylesheets and other external formatting description.

HTML

HTML [73] - HyperText Markup Language - is a markup language derived from SGML and primarily used for web pages. It provides means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, etc. Currently, HTML is the predominant markup language for web pages. In its latest revisions, content and layout are supposed to be separated. Unfortunately common web pages tend to mix both of them.

SVG

Scalable Vector Graphics [35] - SVG - is a specification of an XML-based file format for describing two-dimensional vector graphics, both static and dynamic. SVG has been in development since 1999 by the W3C, it is a standard coming from both PGML (developed from Adobe's PostScript) and VML (developed from Microsoft's RTF). SVG focuses on the graphical rendering, graphical objects can be grouped, styled, transformed, and composited into previously rendered objects. Some SVG features include nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility.

RTF

The Rich Text Format [115] - RTF - is a document file format developed by Microsoft in 1987 for cross-platform document interchange. It is an old proprietary markup language using ASCII characters encoded in an 8-bit format. Most word processing software implementations support RTF format importing and exporting, making it a common format between otherwise incompatible word processing softwares and operating systems. RTF is a useful format for basic formatted text documents such as instruction manuals, resumés, letters, and information documents. RTF supports only a restrictive set of typographical and topological features.

ODA

The Open Document Architecture [50] - ODA - defines a compound document format that can contain raw text, raster images, and vector graphics. ODA documents can express logical as well as layout structures. Logically, the text can be partitioned into chapters, footnotes, and other subelements similar to HTML. It is an ISO standard that addresses the complete range of visual representation issues of how a document looks. ODA is an ambitious standard, its implementations are non-trivial and very few tools actually exist. Moreover, ODA is an ageing file format using outdated and cumbersome encoding schemes that really lack clarity and flexibility.

PDF

Portable Document Format [53] - PDF - is a file format created by Adobe Systems in 1993 for document exchange. It is a simplification and evolution of the Adobe PostScript [2] page description language. PDF is used for representing two-dimensional documents in a manner independent of the application software, hardware, and operating system. Each PDF file encapsulates a complete description of a fixed-layout 2D document that includes the text, fonts, images, and 2D vector graphics which compose the documents.

Nowadays, PDF is a *de facto* file format that has been widely adopted for long term storage and archiving of documents in their electronic form. PDF can be generated from any document processing software in order to get an accurate and fixed-layout representation of an original document. However, PDF focuses on the preservation of the visual appearance of a document and, therefore, does not ensure the preservation of its physical and logical structures. The aim of this thesis is precisely the development of a restructuring workflow for PDF documents, in order to recover their physical and logical structures. Thus, the next chapter, Chapter 4, targets on the strengths and weaknesses of the PDF file format. It proposes a complete solution leading to the reverse engineering of PDF documents.

4

The Pros and Cons of PDF Files

Contents

This chapter is dedicated to the PDF file format, it is divided into seven sections. First, Section 4.1 presents an overview about the PDF format, it exposes the timeline of the different PDF releases and highlights some of its key features. Section 4.2 details the internal structure of a PDF file, as well as the data representation. Section 4.4.1 emphasizes the internal complexity of the file format and, thus, the troubles encountered when reading the content of a PDF file. Thereafter, Section 4.3 underscores the strengths of the PDF file format, while Section 4.4 underlines its weaknesses. Consequently, Section 4.5 explains why users need PDF reverse engineering. Finally, Section 4.6 proposes a complete solution for a reverse engineering workflow.

4.1 PDF History

PDF inherits from the long evolution of page description languages - PDLs -, which been described as "communication of an already formatted page or document description from a composition system to an output device, either screen or printer" [78]. The initial development of PDLs has been closely linked to progress in laser-printer technology: they ensured high-quality print jobs with precise format layout and high resolution images. More recently, PDLs have evolved further so that they are now being used for the electronic dissemination and storage of documents: Adobe's Portable Document Format [53] (PDF) as well as Microsoft's XML Paper Specification [119] (XPS) are precisely two examples.

The PDF file format was first introduced in 1990 as a way to reliably view, print, and share information with other people. PDF was a simplification and evolution of the ageing Adobe PostScript page description language. Whereas Postscript is actually a programming language primarily used to drive laser printers, PDF has been designed specifically for exchanging documents to be rendered on screens. Thus, PDF is an elaborated descriptive format allowing each page to be rendered independently. On the contrary, PostScript needs a dedicated interpreter executing a program and containing a global state for the entire document, i.e., rendering a single page in PostScript imposes the rendering of all the previous pages in order to obtain the current page's state.

Since its beginning, PDF format has greatly advanced, involving progressively a very large set of features (see Figure 4.1). Annotation tools have been added to PDF 1.2 as well as interactive form fields which can be filled in by the user; new accessibility features have also become available for disabled people. Version 1.3 added the support of digital signatures, new color spaces and profiles, and Javascript actions, whereas version 1.4 added the support for OCR text layers. The latter is a real improvement that allows a PDF file, generated from scanned hard copies, to contain both the scanned images and their overlaying and transparent OCRized textual contents. Support for true graphic transparency has also been added in version 1.4.

The rise of internet and its hyperlinked world greatly influenced the PDF version 1.5: linked multimedia were added as well as object streams. Such streams, allowing PDF objects to be encapsulated and compressed, are the roots of PDF inlined documents which enable faster internet transfers thanks to smaller PDF file sizes and inlined transfers, i.e., only the requested parts of a document can be transferred. Security features, that grant document encryption and authentication, were introduced early and progressively improved. The latest PDF release even provides 256-bit AES encryption.

Since the internet boom, PDF emerged as a convenient way to reliably distribute document across the web while preserving the original layout. Thus, in the last decade, PDF became the de facto standard for exchanging electronic documents over the Internet. Thanks to this popularity, PDF writers and export facilities emerged quickly, permitting any kind of electronic document to be output in the Adobe's fixed-layout page description languages. According to Adobe Systems Incorporation [55], more than 200 million PDF documents are currently available on the web. As of today, Adobe has released its version 1.7: a complete PDF Reference (more than 1300 pages) is now available via their web site [53].

4.2 PDF File Format

PDF is an elaborated page description language allowing complex documents containing text, graphics, and images to be unfailingly reproduced, i.e., they will look exactly the same even when rendered on different systems and physical devices (screens, printers).

Version	Year of publication	New features	Supported by Reader version
1.0	1993		Acrobat Reader (Carousel)
1.1	1996	Passwords, device-independent color, threads and links	Acrobat Reader 2.0
1.2	1996	Interactive page elements, mouse events, multimedia types, Unicode, advanced color features and image proxying	Acrobat Reader 3.0
1.3	2000	Digital signatures; ICC and DeviceN color spaces; JavaScript actions	Acrobat Reader 4.0
1.4	2001	JBIG2; transparency; OCR text layer	Acrobat Reader 5.0
1.5	2003	JPEG 2000; linked multimedia; object streams; cross reference streams	Adobe Reader 6.0
1.6	2004	Embedded multimedia; XML forms; AES encryption	Adobe Reader 7.0
1.7	2006		Adobe Reader 8
1.7 Extension Level 3	2008	256-bit AES encryption	Adobe Reader 9

Figure 4.1: PDF version releases and corresponding main new features [114].

This is made possible because PDF integrates the Adobe imaging model, which gives a unified view of two-dimensional graphics, covering vector graphics as well as raster graphics. Indeed, the imaging model is meant to be device independent: text, images, and graphics positions together with their affine transformations are all specified within an internal homogeneous coordinate space.

Since our research concentrates on the restructuring of electronic documents, the following subsections describe in detail the PDF file format and its imaging model in order to provide the basic knowledge needed to understand the next sections.

4.2.1 File Structure and Syntax

The internal structure of a PDF file is decomposed into four primary sections as illustrated in Figure 4.2. The PDF file *header* is just one or two lines starting with "%PDF" and only specifying the current PDF file version. The *body* is the interesting part of the PDF file. It contains the collection of objects that constitute the document. The *xref table*, or cross reference table, is a collection of pointers offering an efficient mechanism to locate the individual objects contained in the body (i.e., random access). The *trailer* encloses the pointer to the start of the cross reference table together with some other special objects within the body of the file. Except for the header, each section may be incrementally updated.

All the content of a PDF document is encapsulated in the body section of the file. It is a flattened representation of a data structure consisting of a set of root *PDF objects*, each of them identified by a unique reference. Actually, the term *object* refers to eight different information types: boolean, number, string, name, array, dictionary, stream, and

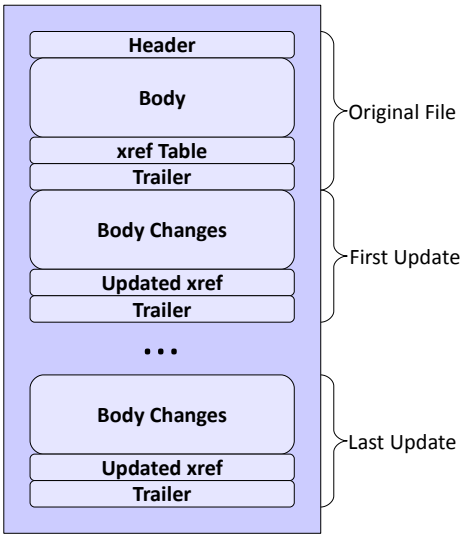


Figure 4.2: The PDF file structure is decomposed into four main sections: a header, a body, an xref table, and a trailer. Except the header, each part may be incrementally updated.

the null object. Composite objects such as arrays, dictionaries, and streams may refer to other objects directly, by embedding them, or indirectly by means of *indirect references*. Since the version 1.5, root PDF objects themselves can be embedded in PDF streams; this enables the possibility to compress or encode such objects in order to save space or encrypt them.

As precited, the body of a PDF file is made of root PDF objects which are themselves expressed using syntactical tokens (and byte streams for stream objects). These *low-level* objects are then combined together to form *high-level* tree structures, thanks to dictionary objects and indirect references, thus allowing all the existing PDF structures to be represented, e.g., pages, document contents, images, fonts, annotations, etc. A PDF document is basically the assembling of all these high-level objects, resulting in a complex tree structure of objects.

A PDF file contains an entry point called a *catalog*, referencing a *page tree* which points to *pages nodes* themselves pointing to *page nodes*. Pages nodes are used to group pages with similar or shared properties, but they can also be used simply to group pages for easier internal navigation, whereas a single page node is used to store the actual content of a specific page. For each page that is displayed in a PDF viewer, there is a corresponding page node within the PDF document.

A page node consists of a set of properties and resources belonging to the current page and one or more content streams. The resources and properties available for a page include fonts, the dimensions of the page, the rotations of the pages, and so on. A content stream

is basically a standard stream object including a specific graphical description language that ensures the exact visual rendering of a PDF page. It is the content streams that, on the one hand, comprises the actual page content and, on the other hand, defines its typographical appearance.

At the lowest level, a PDF file amounts to a sequence of bytes, grouped into syntactical tokens. That is, objects are described by means of tokens, themselves composed of a subset of the printable ASCII set, whereas byte streams are used exclusively in stream objects. Thus, the raw data of a PDF file can be viewed with any plain text editor.

As a basic example, Figure 4.3 introduces the graphical output of a simple PDF file, the corresponding ASCII/binary code is then displayed in Listing 4.1. Let us explain this PDF byte code in more details. A root object is delimited by two tags "obj" and "endobj", and preceded by two numbers, a reference and a generation number. Names and comments are preceded by "/" and "%", respectively. A dictionary object is delimited by the following tags "≪" and "≫", it is composed of a sequence of key-value pairs, i.e., a name followed by some object. An array is a simple list (or sequence) of objects delimited by the following tags "[" and "]". A special construct called "indirect reference" is used to reference a root object: it is constituted of two numbers, a reference and a generation number, directly followed by the character "R". Finally, a stream object is composed of a preliminary dictionary object directly followed by a byte stream which is delimited by the tags "stream" and "endstream".

Figure 4.3: Graphical output of a simple PDF file. This drawing will be used to illustrate various examples along the sections of this chapter.

Listing 4.1: ASCII and binary codes corresponding to the "tiny PDF drawing" example. We observe the four sections of a PDF file, i.e., the header, the body, the xref table, and the trailer.

```
   %PDF-1.4
2 0 obj <</Length 3 0 R/Filter/FlateDecode>> stream
%the binary content stream has been removed since it is not human readable
endstream endobj
3 0 obj 217 endobj
5 0 obj <</Type/Font/Subtype/Type1/BaseFont/Helvetica/Encoding/WinAnsiEncoding
    >> endobj
6 0 obj <</F1 5 0 R>> endobj
7 0 obj <</Font 6 0 R/ProcSet[/PDF/Text]>> endobj
```

```
1 0 obj <</Type/Page/Parent 4 0 R/Resources 7 0 R/MediaBox[0 0 595 842]/Group
    <</S/Transparency/CS/DeviceRGB/I true>>/Contents 2 0 R>> endobj
4 0 obj <</Type/Pages/Resources 7 0 R/MediaBox[ 0 0 595 842 ]/Kids[ 1 0 R ]/
    Count 1>> endobj
8 0 obj <</Type/Catalog/Pages 4 0 R/OpenAction[1 0 R /XYZ null null 0]>>
    endobj
9 0 obj <</Creator<FE FF 00 57 00 72 00 69 00 74 00 65 00 72 > /Producer<FE FF
     00 4F 00 70 00 65 00 6E 00 4F 00 66 00 66 00 69 00 63 00 65 00 2E 00 6F
    00 72 00 67 00 20 00 32 00 2E 00 33> /CreationDate(D:20071203141814+01'00'
    )>> endobj
xref
0 10
0000000000 65535 f
0000000502 00000 n
0000000019 00000 n
0000000307 00000 n
0000000644 00000 n
0000000327 00000 n
0000000418 00000 n
0000000449 00000 n
0000000742 00000 n
0000000825 00000 n
trailer
<</Size 10/Root 8 0 R/Info 9 0 R/ID [ <35285DC804A94B9DCC0087798797448A><35285
    DC804A94B9DCC0087798797448A> ]
/DocChecksum /27D50D004B7C9BEB26D80A52D8EA1303
>>
startxref
1011
%%EOF
```

4.2.2 Content Streams

The graphical content (i.e., text, images, and graphics) of PDF document pages is expressed exclusively inside dedicated PDF streams called content streams. A content stream consists of a sequence of instructions which are used to describe the graphical elements that are to be painted onto the page.

Listing 4.2 precisely shows the decompressed content stream of the binary data stream object exposed in Listing 4.1. Once decompressed, the content stream reveals its sequence of instructions, which are themselves made of operators and operands represented by ASCII characters forming syntactical tokens. A variable amount of operands, represented in the form of PDF objects, and a unique postfixed operator create a single instruction. An instruction can describe either a graphical primitive (text, graphic paths, and raster images) composing the document, or a property used to set or modify a graphics state. References to external PDF objects, i.e., objects not embedded in the content stream, are made possible thanks to operand names mapping objects contained in a page resource dictionary. Note that a special external object called an *XObject* can describe a standalone graphics content in a self-contained content stream and be referenced from another content stream.

The graphics state is a mechanism used in content streams in order to store the actual rendering properties. It is updated by dedicated graphical instructions and can be saved and restored during the rendering of the content stream, using a stack-based mechanism. Each graphical primitive, including character shapes, are drawn accordingly to the current graphics state, which consists basically of fill and draw colors, a clipping path, and a current transformation matrix called CTM. This CTM provides a mapping from the user space coordinates, in which the document is represented, into output device coordinates. The CTM, which is part of the current graphics state, is updated during the rendering of the page thanks to dedicated instructions using relative positioning.

During the rendering phase, the processing order of the graphical primitives corresponds to their z-ordering, i.e., primitives are drawn by following their apparition order. Actually, the page layout of a PDF document is entirely left to the content streams which locate graphical primitives using relative coordinates thanks to the CTM.

Listing 4.2: Decompressed content stream corresponding to the byte codes of Listing 4.1. A content stream is a sequence of instructions themselves composed of an operator preceded by some operands.

```
2 0 obj <</Length 3 0 R/Filter/FlateDecode>> stream
0.1 w
q 0 0.1 595.3 841.9 re W* n
0.8 0.8 1 rg
108 733.4 m
88.1 733.4 72 741.6 72 751.7 c 72 761.8 88.1 770 108 770 c
127.9 770 144 761.8 144 751.7 c 144 741.6 127.9 733.4 108 733.4 c h
f*
q 0 0 0 rg
BT
83.3 754.6 Td /F1 12 Tf[<74>28<69>-28<6E>-11<7920>28<50>-16<44>-28<46>-14<20>]
    TJ
ET
Q
q 0 0 0 rg
BT
86.7 740.4 Td /F1 12 Tf[<64>-11<72>25<61>-11<77>-28<69>-28<6E>-2<67>]TJ
ET
Q
Q
endstream endobj
```

4.2.3 Text Representation

In PDF, the textual content is handled by text instructions defining graphical text primitives embedded in page content streams. These text primitives are determinated by various attributes, including a font id (that maps to a font object held in the page resource dictionary), a font size, some character codes and their relative coordinates on the page.

Most of the time, text is represented as vector graphics and is rendered in the form of a set of character shapes mapping character codes. Since PDF considers glyphs, i.e., character shapes, as graphical objects, many of the text operators are handled using the graphics state and standard painting operators. However, glyphs have their own properties; therefore, in addition to the graphics state, a text state covers the specificities of the text primitives. This text state contains a subset of the graphics state's parameters relevant to the text, coupled with a text matrix and with other additional functionalities for defining fonts, font sizes, and other parameters, e.g., for handling character and word spacing.

Listing 4.3 shows the operators and operands involved in the generation of the text "tiny PDF drawing". We observe that the text primitives are delimited by two tags, "BT" and "ET". The first instruction is a set of two numbers postfixed by the operator "Td" and used to update the current text matrix coordinates. Then, the operator "Tf" selects the current font thanks to the external object named "/F1" and using a font size of 12 points. Finally, the operator "TJ" renders an array of character codes represented by hexadecimal values (delimited by "<" and ">") and adjusted by some inserted position offsets.

Listing 4.3: Some text primitives extracted from the decompressed content stream of Listing 4.2. These text primitives are coded in hexadecimal and form the sentence "tiny PDF drawing".

```
BT
83.3 754.6 Td /F1 12 Tf[<74>28<69>-28<6E>-11<7920>28<50>-16<44>-28<46>-14<20>]
    TJ
ET

BT
86.7 740.4 Td /F1 12 Tf[<64>-11<72>25<61>-11<77>-28<69>-28<6E>-2<67>]TJ
ET
```

Let us note that the textual layout is completely merged with its content, in a way that it does not contain any formatting operator such as line break, paragraph, and indentation. Worst, characters are not necessarily grouped to form words and lines. The textual content is most of the time scattered into meaningless text chunks. Therefore, there is no default mechanism for ascertaining the reading order of content within a traditional PDF file.

4.2.4 Graphics Representation

A graphic, i.e., a vectorial image, is built from a sequence of paths contained in a page content stream. Each path inherits the attributes of the current graphics state and is itself described as a set of consecutive line segments and curves. Paths can be used for painting strokes, filling areas, describing font glyphs, and even clipping zones.

Listing 4.4 shows an extract of the decompressed content stream of Listing 4.2 corresponding to the blue ellipse of the Figure 4.3. The ellipse is composed of four Bezier curves, it is

described using the operators *m*, *c*, and *h* standing for *moveTo*, *cubicCurve*, and *closePath*, respectively. The last instruction is composed of the single operator "f*" asking the PDF renderer to fill the path using the even-odd filling rule.

Listing 4.4: Some graphic primitives extracted from the decompressed content stream of Listing 4.2 and corresponding to the blue ellipse of the Figure 4.3. The resulting shape is an ellipse composed of four cubic curves.

```
108 733.4 m
88.1 733.4 72 741.6 72 751.7 c 72 761.8 88.1 770 108 770 c
127.9 770 144 761.8 144 751.7 c 144 741.6 127.9 733.4 108 733.4 c h
f*
```

4.2.5 Images Representation

Raster images, i.e., digital images, are most of the time embedded in the page resources as stream objects. They are then simply referenced from the content streams by their resource names and positioned (and resized) according to the CTM. Images can also be referenced as external resources, but PDF recommandations discourage such practice since this would link a PDF document with many external files.

Image byte streams are generally compressed using standard compression algorithms such as CCITT for binary images, LZW for synthetized images, or JPEG for photographs. Various color spaces and profiles may be defined in order to control precisely the color rendering scheme. Recent PDF specifications also support alpha blending and image compositing as native PDF operators.

4.2.6 Fonts Representation

PDF is able to deal with many different font formats including the so-called *simple* and *composite* fonts. Simple fonts are limited to 256 glyphs, while composite fonts are unlimited in their glyph space. Since file size is a key issue in PDF, fonts are frequently reduced to subsets, describing only used glyphs. Moreover, fonts are most of the time represented as compact descriptions, post-filtered by a compressing algorithm, again to reduce the overall file size. This leads to quite complicated solutions for representing font information.

Fonts are usually embedded in stream objects and managed by dedicated font programs relative to their format, e.g., TTF - True Type Font - and CFF - Compact Font Format. The mapping between the font glyph indexes and the text primitives character codes is handled by various fonts' encoding mechanisms. Standard ASCII characters composing Latin texts are generally represented using a *Standard Encoding*. The *Expert Encoding* becomes necessary with text documents containing additional characters useful for more sophisticated typography. Non-latin languages provide their own encoding schemes represented as PDF dictionary objects or directly embedded in font programs. Default font encoding may also be

overridden or altered, by defining a different and custom encoding thanks to font dictionaries. Finally, a mapping from font codes to Unicode is frequently available in the font encoding description or in dedicated dictionary objects.

4.2.7 Metadata and Content Structures

PDF provides various mechanisms to incorporate metadata describing its content, such as annotations and structural information. For instance, a PDF annotation associates a file or data such as a note, link, hyperlink, sound, or movie with a location on the page (using the default user space). PDF annotations also provide a way to interact with the user by means of page locations linked with mouse and keyboard events. Listing 4.5 precisely shows an annotation that redirects a PDF viewer to another page when the user clicks on a predefined rectangular region in the current page.

Listing 4.5: This annotation incorporates an action redirecting the user towards a page represented by its PDF indirect reference

```
93 0 obj
  << /Type /Annot
     /Subtype /Link
          /Rect [71 717 190 734]
          /Border [16 16 1]
          /A << /Type /Action
                /S /GoTo
                    /D [3 0 R /FitR -4 399 199 533]
                 >>
  >>
endobj
```

Since PDF 1.3, PDF's logical structure facilities have been added in order to supply mechanisms for incorporating structural information dealing with the document's content. The logical structure of a document is stored separately from its visible content (stored in content streams), thus allowing the ordering and nesting of logical elements to be entirely independent from the order and location of graphics objects on the document's pages. The logical structure is represented by a *structure tree* (using PDF dictionaries) and PDF documents that contain a structure tree are known as structured PDFs. It is the job of the structure tree within a PDF to contain the structure and to point to the content of the document in the correct reading order.

Actually, the tree structure references the content thanks to *marked content* identifiers within each page content streams (i.e., markers inserted into the content streams). Indeed, the content of a PDF page does not have to be in any particular order and can be in totally separate content streams. Marked content is a mechanism used for indicating which content belongs to a given element node in the logical structure tree.

Tagged PDF is a stylized use of structured PDF that appeared in version 1.4. In structured PDF, there are no extra requirements on either content streams or on the logical structure of a document, whereas tagged PDF imposes further requirements that help standard applications such as Acrobat to make more flexible use of the PDF material. Indeed, customized structures cannot be interpreted by standard applications such as Acrobat Reader unless there is a way to indicate the meaning of the custom tags in terms of the layout properties of the document. Thus, tagged PDF is designed to provide basic facilities in three key areas:

1. Re-use and re-purposing of PDF Documents.
2. Reflow and media generalization of PDF Documents.
3. Accessibility of PDF Documents to people with disabilities (especially vision-related disabilities).

It provides the facilities mentionned above by introducing a number of requirements on the content and logical structure in a tagged PDF document. Those are achieved by the three following requirements:

1. A set of standard structure types very similar to the set of HTML tags must be used, i.e., section, paragraph, heading, list, table, etc.
2. Explicit word demarcation is required and the content must appear in reading order within any given content stream.
3. Mappings must be provided to the unicode standard for any fonts that use custom encodings.

If a user wants a custom structure beyond the set of standard structure types, a mechanism known as role mapping is used to map the custom tagset to the standard one. Unfortunately, even though the PDF specification provides a complete set of high-level objects and content stream operators allowing the PDF writers to incorporate documents' logical structures, very few PDF producers take full benefit of such potentialities.

4.3 PDF Strengths

The PDF file format is now the global **standard** for trusted, high fidelity electronic documentation, it is recognized by industries and governments all around the world. It is a universal file format that has been widely adopted for long term storage and archiving of documents in their electronic form, e.g., newspapers archiving, digital libraries, e-books, and so on. For instance, the British Library digital preservation team uses PDF 1.6 along with JPEG2000 and Mets/Alto in order to archive documents over time. In the article entitled "Why PDF is Everywhere" [72], McKinley emphasizes the strengths of the PDF file format for document management and information retrieval.

Nowadays, PDF is a **free and open** file format supported by a freely available reader and by an ever-growing list of creation, viewing, and manipulation tools. Indeed, since Adobe chose to publish the PDF specification, many companies worldwide developed their own PDF-based solutions and tools to enhance PDF standard functionalities. Furthermore, while formerly a proprietary format, PDF was officially released as an open standard in 2008, and published by the ISO - International Organization for Standardization - as ISO/IEC 32000-1:2008. Proper subsets of the PDF file format have also been standardized under ISO such as: (1) PDF/A, for long-term preservation, i.e., for archiving in environments such as corporates, governments, libraries, etc., (2) PDF/X for the printing and graphic arts, (3) PDF/E for exchange of engineering drawings, and (4) PDF/UA (still under development) for universally accessible PDF files and containing logical structure information.

PDF can be considered as a **universal** document format because it is able to represent any kind of printable information including text, drawings, business charts, 3D graphics, and photos. Furthermore, it is a platform-independent file format which is fully portable and can be viewed and printed on any commonly used computer operating systems.

A PDF document reproduces perfectly its original appearance and, therefore, is considered to be a **reliable** and consistent electronic document format. Indeed, PDF provides high-level facilities that enable applications to describe and render text, images, graphics, and fonts accurately and efficiently. Moreover, the imaging model of PDF allows complex imaging systems to embed their own color profiles, ensuring an accurate result when printed.

Recent versions of PDF include new advanced security features such as password protection or digital signatures. In the last years, PDF has been extended with new high-level features, specialized for encapsulating manual annotations, structural information, and other metadata. Thus, PDF is an **evolving** file format that integrates new up-to-date features regularly.

4.4 PDF Weaknesses

Despite the fact that PDF makes possible to embed metadata and structure information about the content, most PDF documents do not make use of such features. Actually, most PDF producers focus only on the preservation of the physical rendering (i.e., the appearance) of documents. As a consequence, many interesting features based on structures are ignored, although they were originally offered by document processing softwares. For instance, document reusability is very limited.

Copy-paste operation of raw text data may fail since the reading order and the segmentation of single characters are not ensured in PDF. In the case of complex multi-column layout, copy-paste operation is not guaranteed to work correctly when the selected text spans over more than one column. Similarly, the selection of multiple consecutive text lines or paragraphs does not always follow the natural reading order. Finally, the PDF internal structure

and representation is complex and thus quite difficult to access (see Subsection 4.4.1). To summarize, PDF weaknesses can be decomposed into the following three categories:

- PDF is an old file format that evolved during many years, thus leading to a **complex file format** (Subsection 4.4.1).
- PDF does not ensure the preservation of the physical structure and, thus, is subject to **physical structure flaws** (Subsection 4.4.2).
- PDF does not ensure the preservation of the logical structure and is, therefore, subject to **logical structure loss** (Subsection 4.4.3).

4.4.1 Format Complexity

PDF is an "old" file format forced to evolve in order to suit the rapid development of computers, software, and desktop publishing. Since its creation in 1993, PDF file format has been subject to several releases, progressively increasing the number of operators and high-level objects needed to provide new features and up-to-date solutions for document representations. Thus, new operators and object structures were added while old ones were deprecated (but still effective), in order to guarantee back-compatibility with previous PDF readers (that is not always the case). As example, Figure 4.4 shows the evolution of the PDF "action types" carried through the different versions.

PDF file format is rather complex and a given document representation is not unique since different operators and structures can lead to the exact same rendering. The information held in a PDF file may be described using concurrent structures, hence leading to redundant or overriding data. The evolution of PDF file format results in a wide range support of encoding schemes used to represent and compress the PDF data, for instance:

- Images support *ASCIIHex*, *ASCII85*, *LZW*, *Flate*, *Run-length*, *CCITT Fax*, *DCT*, *JBIG*, *JPX* encoding schemes.
- Color spaces support *DeviceGray*, *DeviceRGB*, *DeviceCMYK*, *CalGray*, *CalRGB*, *Lab*, *ICCBased*, *Indexed*, *Pattern*, *Separation*, *DeviceN* encoding schemes.
- Fonts support *Composite*, *Type 1*, *Multiple Master*, *Type 3*, *TrueType*, *CID Type 0*, *CID type 2* encoding schemes.
- Character sets support *Standard*, *MacRoman*, *WinAnsi*, *PDFDoc*, *MacExpert*, *Symbol*, *ZapfDingbats* encoding schemes.

Streams and strings are often compressed and even encrypted in order to protect the PDF contents from unauthorized access. Such filtered streams and strings have to be decompressed and decrypted before their reading. PDF even allows unknown data file formats to be embedded, such as movies and sound files. The reading of these files is then left to the operating system, which may, or may not, be able to open the embedded data.

TABLE 8.44 Action types

ACTION TYPE	DESCRIPTION	DISCUSSED IN SECTION
GoTo	Go to a destination in the current document.	"Go-To Actions" on page 616
GoToR	("Go-to remote") Go to a destination in another document.	"Remote Go-To Actions" on page 617
GoToE	("Go-to embedded"; *PDF 1.6*) Go to a destination in an embedded file.	"Embedded Go-To Actions" on page 617
Launch	Launch an application, usually to open a file.	"Launch Actions" on page 621
Thread	Begin reading an article thread.	"Thread Actions" on page 623
URI	Resolve a uniform resource identifier.	"URI Actions" on page 624
Sound	*(PDF 1.2)* Play a sound.	"Sound Actions" on page 625
Movie	*(PDF 1.2)* Play a movie.	"Movie Actions" on page 626
Hide	*(PDF 1.2)* Set an annotation's Hidden flag.	"Hide Actions" on page 627
Named	*(PDF 1.2)* Execute an action predefined by the viewer application.	"Named Actions" on page 628
SubmitForm	*(PDF 1.2)* Send data to a uniform resource locator.	"Submit-Form Actions" on page 662
ResetForm	*(PDF 1.2)* Set fields to their default values.	"Reset-Form Actions" on page 666
ImportData	*(PDF 1.2)* Import field values from a file.	"Import-Data Actions" on page 667
JavaScript	*(PDF 1.3)* Execute a JavaScript script.	"JavaScript Actions" on page 668
SetOCGState	*(PDF 1.5)* Set the states of optional content groups.	"Set-OCG-State Actions" on page 629
Rendition	*(PDF 1.5)* Controls the playing of multimedia content.	"Rendition Actions" on page 630
Trans	*(PDF 1.5)* Updates the display of a document, using a transition dictionary.	"Transition Actions" on page 632
GoTo3DView	*(PDF 1.6)* Set the current view of a 3D annotation	"Go-To-3D-View Actions" on page 632

Note: *Previous versions of the PDF specification described an action type known as the* set-state action*; this type of action is now considered obsolete and its use is no longer recommended. An additional action type, the* no-op action*, was defined in PDF 1.2 but never implemented; it is no longer defined and should be ignored.*

Figure 4.4: This figure, extracted from the PDF reference 1.6 [53], shows clearly the evolution of the format and, in particular, the available "action types" throughout the PDF revisions.

4.4.2 Physical Structure Flaws

The traditional PDF file format (i.e., non-structured) only ensures the preservation of the visual appearance; there is consequently no guarantee about the order of the text within a content stream. Ergo, the fundamental strength of PDF, i.e., its accurate rendering, is also responsible of some of its worst weaknesses.

As mentioned above, in PDF, the reading order of the text is most of the time not preserved; no assumption can then be made about the order in which character strings appear in content streams. Since there is no requirement about the reading order in the text content, it often appears that, for multi-column layout, the text is rendered across the columns, i.e., by hopping across the inter-column gutter. Moreover, when a PDF file is produced from a typesetting program, it is common to find content streams lacking of space characters. This results from the kerning and from the hyphenation-justification algorithms employed for flowing text onto a page. Actually, the space between words and even between characters is relatively fluid, so instead of using hard-space characters, there is a tendency to use PDF movement operators to position individual characters, or whole words, at the correct position on the page. Thus, depending on the PDF writer, the text can be segmented a word at a time, a character at a time, or by some other segmentation induced by an algorithm potentially trying to optimise on-screen rendering.

Figure 4.5: These various text extracts show the unpredictability of the internal text segmentation of PDF documents.

Figure 4.5 shows the content stream segmentation of the text primitives found in three newspapers. The Swiss newspaper "La Liberté" is segmented into lines that are split; each time a kerning occurs; on the contrary, the "International Herald Tribune" (IHT) is segmented into words that may also be affected by character kernings and thus split. The French newspaper "Le Monde" is also segmented into words subject to kernings, however, white spaces are non-existent. Figure 4.5 reflects only the text segmentation issue, nevertheless, the reading order of these text primitives in the content stream itself is not guaranteed.

Figure 4.6 shows the poor segmentation of the word "Europe" in the content stream. Indeed, "Europe" is both under- and over-segmented, it is composed of the following text

Figure 4.6: The under- and over-segmented word “Europe” extracted from the IHT.

primitives: “E”, “u”, “ ”, “rope””. Moreover, we observe the presence of an undesirable white space which is visually undetectable since it is overlayed by another text primitive “rope””.

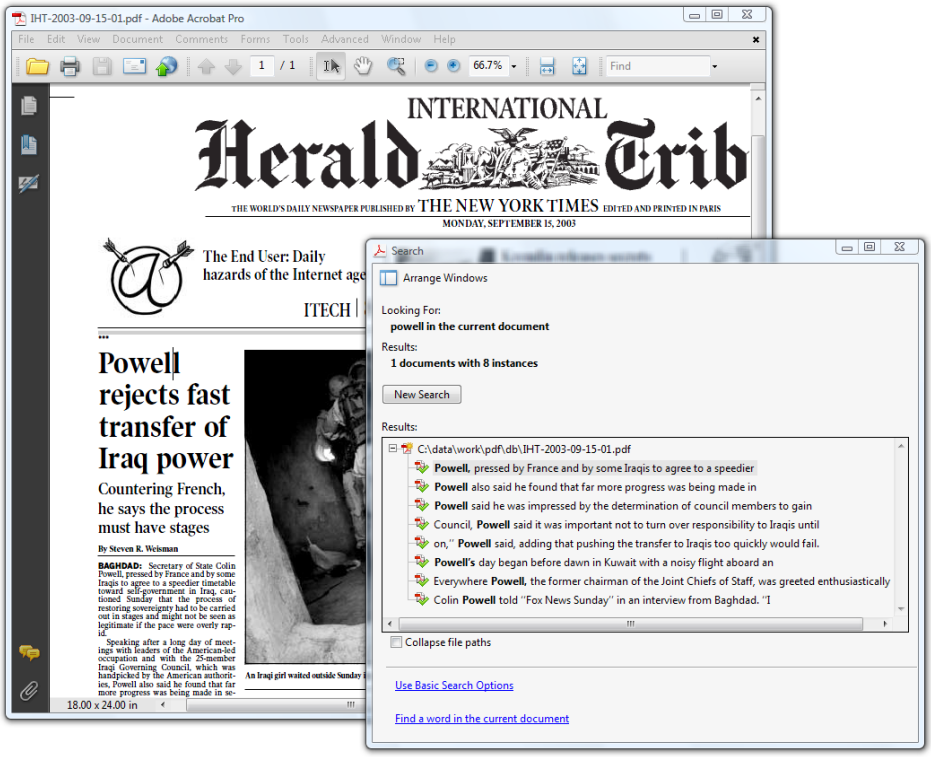


Figure 4.7: An excerpt from the “IHT” showing the over-segmented word “Powell” that jeopardizes text searching.

Similarly, Figure 4.7 shows a screenshot of an article on the left and the results of a search on the word “Powell” with Acrobat Reader on the right. Acrobat simply ignores the word “Powell” in the title, because, internally, it is composed of two sub-strings “Powel ” and “l”, with a blank at the end of the first character string. This inconsistency is not visible to the reader, because during the PDF rendering the character “l” overlaps the blank space of “Powel ”.

Let us note that some PDF applications and readers, such as Adobe Reader 9, do a rough on-the-fly reordering and segmentation of the text primitives contained in the content streams

before accessing them. This processing is especially useful when a user tries to copy or select some text from a PDF file thanks to the graphical interface. Even though Acrobat Reader applies a straightforward word segmentation algorithm, we observed that a search operation on the string "Powell" did not match because of the extra white space.

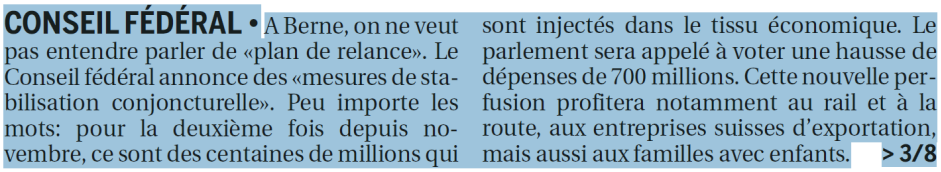
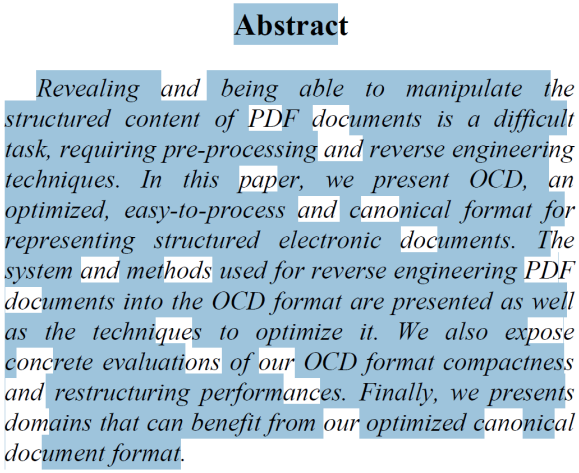

CONSEIL FÉDÉRAL • A Berne, on ne veut pas entendre parler de «plan de relance». Le Conseil fédéral annonce des «mesures de stabilisation conjoncturelle». Peu importe les mots: pour la deuxième fois depuis novembre, ce sont des centaines de millions qui sont injectés dans le tissu économique. Le parlement sera appelé à voter une hausse de dépenses de 700 millions. Cette nouvelle perfusion profitera notamment au rail et à la route, aux entreprises suisses d'exportation, mais aussi aux familles avec enfants. **> 3/8**

Figure 4.8: An excerpt from "La Liberté" showing the selection of an under-segmented text block.

Another great issue with the PDF format is the non-respect of columns which frequently suffer from under-segmentation. Figure 4.8 shows a concrete example of columns under-segmentation. We observe that the segmentation in column is broken down, a line of text is now composed of two distinct text lines coming from adjacent columns and merged together by adding a whitespace between them. A multi-column under-segmentation inevitably implies the reading order to be broken, thus leading to unmeaningful text lines.


Abstract

Revealing and being able to manipulate the structured content of PDF documents is a difficult task, requiring pre-processing and reverse engineering techniques. In this paper, we present OCD, an optimized, easy-to-process and canonical format for representing structured electronic documents. The system and methods used for reverse engineering PDF documents into the OCD format are presented as well as the techniques to optimize it. We also expose concrete evaluations of our OCD format compactness and restructuring performances. Finally, we presents domains that can benefit from our optimized canonical document format.

Figure 4.9: An excerpt from a scientific paper showing an anarchical text selection.

In some cases, the straightforward copy/paste of PDF textual content may be totally unusable due to anarchical internal text primitive ordering. Figure 4.9 exposes such a case, where the manual selection of text lines under Adobe Reader is litteraly impossible.

Last but not least, WYSIWYG - What You See Is What You Get - text editors offer the possibility to add text effects such as shadow and relief. When printing such document into

PDF, these effects may be achieved by writing the text twice and by using distinct colors, intensities, coordinates, and z-ordering (see Figure 4.10).

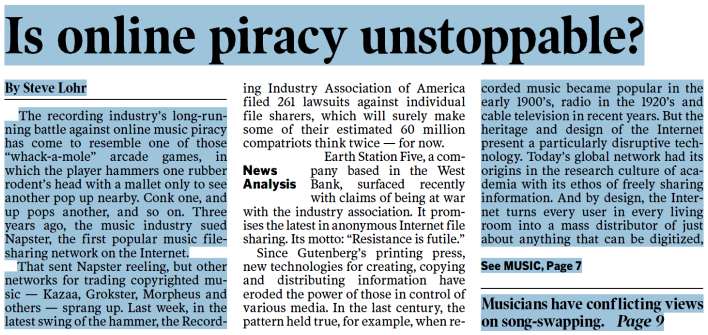
Figure 4.10: An example showing overlapping textual content.

For instance, titles sometimes use this trick and, more generally, newspaper and magazine charts or graphics. The main drawback of this technique is that we finally get the text written twice in the PDF source file. Therefore, a dedicated analysis process will have to detect such text overlapping situations in order to infer the correct text sequences.

4.4.3 Logical Structure Loss

Logical structure loss basically means that the reading order between some physical text blocks (title, subtitle, paragraph, header, footer, etc.) is not respected. This problem is generalized in PDF files and cannot be solved without a deep physical layout analysis followed by a logical structure reconstruction.

Although the PDF file format provides means to embed the logical structure of a document (see Subsection 4.2.7), most PDF writers do not take into account these structures, they are just left aside. Furthermore, lots of PDFs are generated from source files that simply do not contain any logical information: either the user did not use the editor correctly (using word processing editor stylesheets) or the editor itself did not provide such functionalities. Concerning the correct use of word processing editors, users commonly produce their electronic documents with WYSIWYG editors by manually editing the layout and therefore missing stylistic and typographic capabilities of the interfaces. Consequently, content and presentation are often mixed.



Is online piracy unstoppable?

By Steve Lohr

The recording industry's long-running battle against online music piracy has come to resemble one of those "whack-a-mole" arcade games, in which the player hammers one rubber rodent's head with a mallet only to see another pop up nearby. Conk one, and up pops another, and so on. Three years ago, the music industry sued Napster, the first popular music file-sharing network on the Internet.

That sent Napster reeling, but other networks for trading copyrighted music — Kazaa, Grokster, Morpheus and others — sprang up. Last week, in the latest swing of the hammer, the Recording Industry Association of America filed 261 lawsuits against individual file sharers, which will surely make some of their estimated 60 million compatriots think twice — for now.

News Analysis

Earth Station Five, a company based in the West Bank, surfaced recently with claims of being at war with the industry association. It promises the latest in anonymous Internet file sharing. Its motto: "Resistance is futile."

Since Gutenberg's printing press, new technologies for creating, copying and distributing information have eroded the power of those in control of various media. In the last century, the pattern held true, for example, when recorded music became popular in the early 1900's, radio in the 1920's and cable television in recent years. But the heritage and design of the Internet present a particularly disruptive technology. Today's global network had its origins in the research culture of academia with its ethos of freely sharing information. And by design, the Internet turns every user in every living room into a mass distributor of just about anything that can be digitized,

See MUSIC, Page 7

Musicians have conflicting views on song-swapping. *Page 9*

Figure 4.11: An excerpt from the "IHT" showing some text blocks sequence inconsistencies while trying to select an entire article.

Our experience shows that PDF files with complex layouts usually have unpredictable text blocks sequences. Figure 4.11 exposes how the sequential consistency between text blocks may be broken down. Indeed, when trying to select an entire article in Adobe Reader, we observed that the second text block was left aside while the third one was selected right after the first one. A bigger surprise popped out when we looked at the whole newspaper's frontpage while selecting the article. Indeed, Figure 4.13 shows that the article title is itself separated from the rest of the article. That is, during the selection of the article, a lot of other article titles and paragraphs were interleaved.


10 LA LIBERTÉ
MERCREDI 2 MARS 2005

GROS PLAN

Les démocrates-chrétiens ont choisi Thérèse Meyer pour succéder à Jean-Philippe Maitre.

LE PERCHOIR PROMIS À UNE FRIBOURGEOISE

Thérèse Meyer sera élue, la semaine prochaine, présidente du Conseil national. Ceux qui la côtoient estiment que son sens du compromis fera merveille. Elle-même assure qu'elle continuera de défendre ses convictions.

CHRISTIANE IMSAND

Le poste a une forte consonante honorifique. On comprendra que la Fribourgeoise Thérèse Meyer rayonne de satisfaction après sa désignation par le groupe parlementaire démocrate-chrétien pour la succession de Jean-Philippe Maitre au perchoir du Conseil national. Cette nomination qui doit encore être avalisée par le plénum la semaine prochaine – la date précise n'est pas encore connue – récompense une politicienne qui a su se profiler rapidement dans le domaine de la politique sociale et familiale dès son entrée au parlement. Elle ne fait pourtant pas partie des vétérans puisque son élection remonte à 1999 seulement.

L'ancienne syndique d'Estavayer-le-Lac a participé activement à l'élaboration de solutions de compromis dans le domaine du 2e pilier (abaissement du seuil d'accès) et du congé-maternité. Elle s'engage actuellement pour une réduction des primes d'assurance-maladie des enfants et plaide pour la flexibilisation de l'âge de la retraite. C'est par ailleurs une démocrate-chrétienne aux positions classiques qui a milité par exemple contre la dépénalisation de l'avortement.

Pourquoi vous êtes-vous portée candidate?

Thérèse Meyer: – Avant Jean-Philippe Maitre, il y avait plus de vingt ans que le poste n'avait plus été occupé par un PDC latin. J'ai accepté de me présenter pour assurer cette continuité latine. Nous avons cependant décidé avec le parti de renoncer après l'élection à la traditionnelle réception dans les couloirs du Palais fédéral. La fête a déjà eu lieu. Elle s'adressait à Jean-Philippe Maitre.

Vous êtes une femme. Pensez-vous que votre désignation soit aussi une compensation à l'éviction de Ruth Metzler?

– Je n'ai pas eu cette impression. Je suis avant tout la doyenne du groupe latin du Conseil national. Cela a joué un rôle déterminant.

Vous allez être très occupée durant ces prochains mois. Cela pose le problème de la professionnalisation rampante du parlement?

– J'ai la chance de pouvoir me libérer tout de suite et la charge supplémentaire sera limitée à huit mois en ce qui me concerne, puisque le prochain président sera élu en décembre. Quoi qu'il en soit, je reste favorable à un parlement de milicar je crois que quand on est engagé dans des activités de la société civile on sent mieux les besoins et les problèmes de la population.

Vous êtes très active dans les questions de politique sociale. La présidence du National ne va-t-elle pas vous faire perdre en visibilité?

– Je ne pourrai plus monter au pupitre, mais cela ne m'empêchera pas de continuer mon action au sein de la Commission de la sécurité sociale et de la santé publique. Les impulsions sont données et le sacrifice ne sera pas trop lourd.

Pensez-vous profiter de la situation pour faire passer des messages politiques?

– Le président doit avant tout garantir le bon déroulement des débats. Mais pour autant que je sois élue, je ne manquerai pas de rappeler dans mon discours inaugural mes convictions dans le domaine de la politique sociale et familiale.

Propos recueillis par CI

Figure 4.12: An excerpt from "La Liberté" showing a hierarchical structure inconsistency, the head part following the title is not considered as being in the same logical block.

The non-respect of logical hierarchies is also a recurrent logical structure issue in PDF. This happens when a sequence of text blocks do not fit the logical hierarchy of an article (title, subtitle, content, author name, etc.). Figure 4.12 shows the attempt of selecting an entire article with the following result: the selection of the article body was correct, but it directly followed the article title while omitting the head part. Then, when trying to select the head part, the rest of the article was left out. Such logical issues are quite common in PDF and resolving them requires the recovering of the complete logical structure.

Actually, PDF viewers themselves confirm the fact that the logical structure is absent. Such high-level information would otherwise be made available to the user through the interface, e.g., with a tree structure containing an article node.

There is a last interesting point concerning the logical structure inconsistencies while doing a text selection in Adobe Reader. We observed that the text blocks order during the selection sometimes tends to bring together blocks having the same logical function, i.e., titles together, captions together, and so on. This issue certainly results from some PDF writers

INTERNATIONAL
Herald Tribune
THE WORLD'S DAILY NEWSPAPER PUBLISHED BY THE NEW YORK TIMES EDITED AND PRINTED IN PARIS
MONDAY, SEPTEMBER 15, 2003

The End User: Daily hazards of the Internet age ITECH | 8
Kremlin releases secrets on Khrushchev's blustering FOCUS | 2
Will 25 be too many at EU's new table? A SPECIAL REPORT | 11

Powell rejects fast transfer of Iraq power

Countering French, he says the process must have stages

By Steven R. Weisman

BAGHDAD: Secretary of State Colin Powell, pressed by France and by some Iraqis to agree to a speedier timetable toward self-government in Iraq, cautioned Sunday that the process of restoring sovereignty had to be carried out in stages and might not be seen as legitimate if the pace were overly rapid.

Speaking after a long day of meetings with leaders of the American-led occupation and with the 25-member Iraqi Governing Council, which was handpicked by the American authorities, Powell also said he found that far more progress was being made in securing and rebuilding Iraq than had been indicated in news reports.

Powell said he was impressed by the determination of council members to gain control of their country as quickly as possible, and he said the United States fully supported them. "This is how you build a government," the secretary said, describing a process in which Iraqis take on "more and more responsibilities" over time.

Refusing to set an exact time for this process as sought by France and the United Nations Security Council, Powell said it was important not to turn over responsibility to Iraqis until the government is seen as legitimate as a result of a new constitution and elections, which could take place well into next year.

"We're not hanging on for the sake of hanging on," Powell said, adding that pushing the transfer to Iraqis too quickly would fail. "The worst thing that could happen is for us to push this process too quickly," he said.

Powell's day began before dawn in Kuwait with a noisy flight aboard an Air Force transport plane to Baghdad airport and then a helicopter ride to the Presidential Palace on a bend in the Tigris River that serves as occupation headquarters. Everywhere Powell, the former chairman of the Joint Chiefs of Staff, was greeted enthusiastically by servicemen and women who seemed to regard him as one of their own.

He traveled through a hot, dusty and brown city that has been returning to normal but one in which American compounds are walled off, fortified and isolated because of the bombing attacks since mid-summer.

As for the timetable for democracy, it is a matter of much discussion, notably because of France's strong criticism and the desire by some Security Council members to speed things up in order to win France's assent to a resolution to widen the UN's role over security

See IRAQ, Page 7

Top Bush aides reinforce defense of conduct in Iraq. *Page 5*

Newsstand prices

France € 1.85

Algeria ..... Dn 40 Ivory Coast ... CFA 1,900
Andorra ..... € 1.85 Reunion/Antilles ... € 2.20
Cameroon ... CFA 1,200 Senegal ..... CFA 1,900
Gabon ..... CFA 1,500 Tunisia ..... Din 2.500

For information on delivery, or to subscribe in France, call toll-free:
00 800 44 48 78 27

Ariko Datta/Reuters

An Iraqi girl waited outside Sunday in a village north of Tikrit as American soldiers searched her house, looking for armed robbers.

Voters in Sweden say 'no' to euro

56%-to-42% outcome is blow to Europe

By Alan Cowell

STOCKHOLM: After a passionate campaign made uncertain to the last moment by the assassination of Foreign Minister Anna Lindh, Swedes voted overwhelmingly in a referendum to reject membership in the European single currency, according to final results tallied late Sunday.

The margin exceeded the expectations of many people here and provided a stunning defeat for Prime Minister Goran Persson. Analysts said the impact would be felt across Europe, countering efforts to bolster the 15-nation European Union before its planned expansion with 10 new members next year. Apart from Sweden, Denmark and Britain are currently outside the 12-country euro zone stretching from the Arctic Circle to the Mediterranean.

With ballots tallied from virtually all 5,976 voting districts, the anti-euro campaign secured victory by a margin of about 56 percent to 42 percent. The result suggested that Swedes had not been swayed by the killing of Lindh, thereby avoiding a sympathy vote in favor of the single currency.

Lindh was a fervent campaigner in favor of adopting the euro in place of the Swedish krona. She died of stab wounds after an unidentified assailant attacked her as she shopped in a department store in central Stockholm last Wednesday.

Shortly before the final tally Sunday night, Persson appeared before reporters to acknowledge the defeat of the "yes" campaign.

"In this result we can see a deep skepticism toward the entire euro project among the Swedish people," he said. "It was a clear expression of the people's will."

Analysts have always calculated that a "no" vote in Sweden will strengthen anti-euro sentiment in Denmark and Britain. Denmark rejected adoption of the euro in a referendum three years ago. Britain has yet to decide when to hold a vote among its people. The Swedish government has said that it will not hold another vote on joining the euro for 10 years.

The campaign leading up to the referendum Sunday had threatened to be overwhelmed by the death of Lindh, whose killing stunned Sweden, rekindling memories of the 1986 slaying of Prime Minister Olof Palme. The police said Sunday they had not arrested a suspect in the killing of Lindh.

Until Lindh's death, opinion surveys had shown the "no" campaign firmly ahead by 10 to 15 percentage points. That margin seemed to have survived intact despite a last-minute flurry of opinion surveys suggesting a "yes" vote was still possible.

"There's a bigger fear for the new than we expected," said Ulrika Messing, a government minister.

The debate about the euro has been so fractious that five members of Persson's government spoke out against it, dividing the Social Demo-

See VOTE, Page 7

Is online piracy unstoppable?

By Steve Lohr

The recording industry's long-running battle against online music piracy has come to resemble one of those "whack-a-mole" arcade games, in which the player hammers one rubber rodent's head with a mallet only to see another pop up nearby. Conk one, and up pops another, and so on. Three years ago, the music industry sued Napster, the first popular music file-sharing network on the Internet.

That sent Napster reeling, but other networks for trading copyrighted music — Kazaa, Grokster, Morpheus and others — sprang up. Last week, in the latest swing of the hammer, the Recording Industry Association of America filed 261 lawsuits against individual file sharers, which will surely make some of their estimated 60 million compatriots think twice — for now.

News Analysis

Earth Station Five, a company based in the West Bank, surfaced recently with claims of being at war with the industry association. It promises the latest in anonymous Internet file sharing. Its motto: "Resistance is futile."

Since Gutenberg's printing press, new technologies for creating, copying and distributing information have eroded the power of those in control of various media. In the last century, the pattern held true, for example, when recorded music became popular in the early 1900's, radio in the 1920's and cable television in recent years. But the heritage and design of the Internet present a particularly disruptive technology. Today's global network had its origins in the research culture of academia with its ethos of freely sharing information. And by design, the Internet turns every user in every living room into a mass distributor of just about anything that can be digitized,

See MUSIC, Page 7

Musicians have conflicting views on song-swapping. *Page 9*

Behind vote, a mistrust of Germany and France

By John Vinocur

Sweden's rejection of the European Union's common currency was a coldly rational statement of mistrust in the euro. It distanced itself from the stagnant economic performance of the euro zone, and flatly refused to place its monetary future under the sway of big countries like France and Germany, who have bent the euro's rules to suit themselves.

News Analysis

Coming at a moment when emotions after the murder of Foreign Minister Anna Lindh might well have pointed the electorate in the direction of the "yes" vote she favored, the "no" in the referendum Sunday proposing adoption of the euro wasn't just a slap, but a hard right, to the jaw of European monetary union.

As a country that believes in rules and reasonableness, that treasures moderation, and bases its economic life on exports and solid international relations, Sweden said, in effect, that it could do better on its own.

Despite its shock and grief over the death of its pro-euro foreign minister, Sweden restrained its emotions and said it was not joining a currency that had provided more signs of self-interested maneuvering among its strongest members than a willingness to play by the regulations they conceived themselves.

For the euro's participants, and for those outside Europe, it was an embarrassing rejection. It made euro-Europe look like no place to go for an enlightened electorate — after the Danes' and Swedes' refusals, only Britain has a referendum on its schedule — and euroland like mostly a split-level club for the noncomplying founder countries and the needy new members from the continent's east and south.

In the weeks before the vote, Prime Minister Goran Persson, a euro supporter, had said that a substantial source of opposition to joining the currency resulted from France's and Germany's unwillingness to meet the euro zone's annual deficit criteria — a reference, in France's case, to its prime minister's statement that its European obligations would always be subordinate to France's self-interest.

After Lindh's death, Finance Minis-

See SWEDES, Page 7

Estonians approve EU entry, early results show. *Page 7*

UPDATE

Rich-poor gap sinks trade talks

Talks that would have changed the way farm products around the world are traded collapsed Sunday amid differences between rich and poor nations, delegates to the World Trade Organization said in Cancún, Mexico.

It was the second major defeat for the WTO, the body that makes rules for international trade. In 1999, similar talks collapsed in Seattle amid street riots and difference between members.

The issues on the table Sunday included foreign investment and competition. *Page 9*

On the Web: *www.iht.com*

Israel won't rule out killing Arafat

By Greg Myre

RAMALLAH, West Bank: Israel's deputy prime minister said Sunday that killing the Palestinian leader, Yasser Arafat, was one of several options now under government consideration. Meanwhile, a beaming Arafat soaked up the cheers of supporters who descended on his compound for a fourth straight day.

"Arafat can no longer be a factor in what happens here," Deputy Prime Minister Ehud Olmert told Israel radio.

"The question is: How are we going to do it? Expulsion is certainly one of the options, and killing is also one of the options," he said. "In my eyes, from a moral point of view, this is no different than the eliminations of others who were involved in activating acts of terror."

Israel has carried out dozens of killings of Palestinian militants during the last three years of fighting.

Olmert said Israel could also leave Arafat confined to his West Bank offices in Ramallah, but cut his communications and bar visitors.

Israel has given no indication of when, or if, it might act to oust Arafat. Prime Minister Ariel Sharon's government is facing blanket opposition from other nations, including the United States, Israel's strongest ally.

"The United States does not support either the elimination of him or the exile of Mr. Arafat," Secretary of State Colin Powell told "Fox News Sunday" in an interview from Baghdad. "I think you can anticipate that there would be rage throughout the Arab world, the Muslim world, and in many other parts of the world."

Arafat's badly damaged compound has often seemed listless and semi-deserted during his confinement over the last year and a half. But since Israel's announcement Thursday night that it had decided in principle to remove Arafat, the complex has taken on a rollicking carnival atmosphere.

On Sunday, a marching band thundered and Palestinian folk dancers took to a stage set up on the black-

See MIDEAST, Page 7

Backers' cheers buoy Palestinian leader

No stunt unspurned

Bored Britons taunt dangling illusionist

By Sarah Lyall

LONDON: To the extent that there is a consensus among those watching the illusionist David Blaine's latest endurance feat, it is that he did not deserve to be pelted with eggs, attacked by golf balls, harassed with laser pens, or jolted awake by people banging on drums, as has been his lot since Sept. 5.

But that does not mean that Blaine's latest stunt — living in a see-through box suspended from a crane near Tower Bridge — is winning many new fans among the hard-to-please Britons gathered below.

"It's rather pointless, I must say," remarked Philip Landau, a 38-year-old lawyer, watching Blaine lie down and then get up again. "There's not much to watch. It's like watching someone sitting in a box."

Blaine entered the plastic box, measuring seven feet high, seven feet long and three feet wide, on Sept. 5, vowing to spend 44 days there with no food and only water to drink. "The feeling of wonder is amazing," he said.

In the past, Blaine's death-defying acts have included standing for a long time on a pedestal, burying himself alive in a glass coffin and encasing himself in a massive block of ice. But never before has he had to confront the particular brand of British cynicism that regards him, as Jane Moore wrote

See BOX, Page 7

Max Nash/The Associated Press

Suspended in a box over London, David Blaine hopes to endure 44 days without food.

ADVERTISEMENT

CONRAD HOTELS
Earn triple airline miles! Book any one of Conrad's luxury hotels online at ConradHotels.com stay before 31st December 2003 and earn triple airline miles*
Bangkok · Brisbane · Brussels · Cairo · Dublin · Gold Coast · Hong Kong · Hurghada · Istanbul · Kilkenny · London · New York · Punta del Este · Sharm El Sheikh · Singapore
ConradHotels.com
Promotion only available when booking online at www.ConradHotels.com and is not available in conjunction with any other offer. Hilton HHonors membership, earning of Points and Miles™, and redemption of points subject to HHonors Terms and Conditions.

ADVERTISEMENT

Figure 4.13: The "IHT" frontpage showing the article selection issue: the logical structure of the document is not respected at all.

that first output text primitives having similar properties.

4.5 Why do Users Need PDF Reverse Engineering?

Nowadays, a huge amount of PDF files still lack physical and logical structures. This is mainly due to three factors: (1) there is a lack of document production systems for creating documents with customised embedded logical structure, (2) a vast quantity of PDF documents were created even before PDF logical structure was available, and finally, (3) many PDF documents are now generated from OCRized paper documents.

The lack of physical and logical structures has several drawbacks when using and processing PDF documents. For instance, PDF documents can not be reedited, restyled, or reflowed easily, i.e., it is not possible to modify the content of a document, to change its general presentation, or to adapt another style. Even simple copy-paste operations do not ensure the preservation of the original physical and logical information; thus, reusing text excerpts of existing PDF documents requires human intervention in order to check the extracted text content and then do the restyling manually.

Accessibility is also a key feature of current electronic documents. For instance, screen readers and other adaptive equipment used by disabled people, i.e., synthetic speech or Braille ouptut systems, precisely need logically tagged PDF files. Similarly, portable devices such as PDA - personal digital assistant - and smartphones such as Apple's iPhone and Google's Nexus One phone need PDF documents to be reflowed in order to fit their low screen resolution (320 x 480 and 480 x 800 pixels, respectively).

Other potential applications relate to information indexing and retrieval. Currently, PDF databases use poor indexation systems that handle PDF files as simple bags of words. But information indexing and retrieval would greatly benefit from document logical structures. For instance, it would be possible for a user to retrieve only the titles of documents or caption of figures. Database systems should use the logical structure information in their indexing tables in order to improve the user queries.

Systems and users working with PDFs need full access to their structures. However, none of the existing restructuring systems provide a complete and interactive analysis workflow (see Chapter 3). There is a need for new electronic document analysis systems, able to restructure PDF documents thanks to assisted, dynamic, and friendly interfaces. Such environments should provide elaborated recognition systems improving with use thanks to incremental learning capabilities.

4.6 Proposed Solution

During this thesis, solutions have been investigated in order to overcome PDF limitations. It resulted in two complementary restructuring systems, called *XED* and *Dolores*, which aim

at extracting respectively the hidden physical layout and the logical structural information from PDF documents. Our final goal is to convert a PDF file into a structured format, still preserving the layout, and further enabling to reapply all kinds of editing operations offered by common text processing systems.

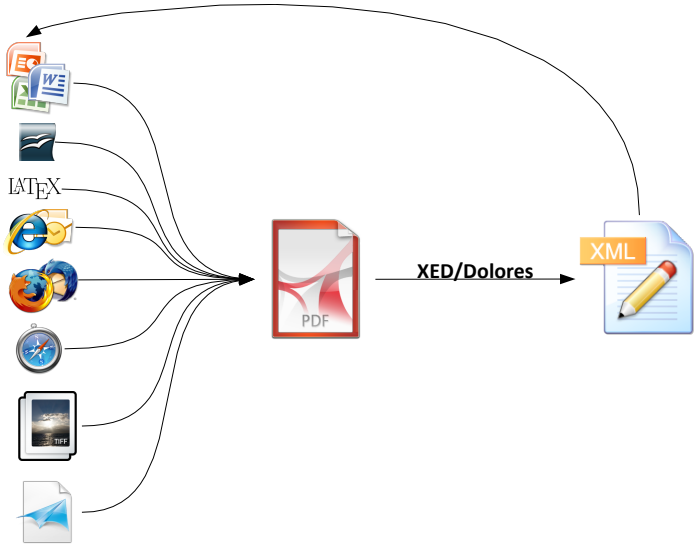


Figure 4.14: Re-enabling the PDF life cycle.

The power of our combined system, XED/Dolores, resides in its ability to re-enable the life cycle of printed documents, as shown on Figure 4.14. PDF is a "dead end" format that can be generated from any kind of electronic document (Word and Presentation processors, HTML, Latex, etc.) as well as paper documents (thanks to OCRs). Thus, we consider the restructuring task as a kind of reverse-engineering performance, in the sense that it opens the possibility of regaining, or even gaining in some cases, the logical structure information from the physical rendering.

As described in the following chapters, PDF reverse-engineering is accomplished by first extracting the entire document content, then recovering its associated physical structure (Chapter 5), and finally reconstructing the lost logical structure (Chapter 7). XED and Dolores are precisely our two distinctive tools that have been developped in order to assume these required restructuration steps (see Figure 4.15).

XED - eXploring Electronic Document - is a tool that aims at recovering the physical structure from PDF documents. XED converts the original PDF document into an intermediate representation called OCD - Optimized Canonical Document - describing the physical structure of the document while preserving its exact visual rendering. Since the physical structure can be defined universally, this initial step is supposed to be generic and applicable on any document class containing textual and graphical content. The process leading to our OCD canonical format involves many steps such as PDF parsing, PDF virtual document

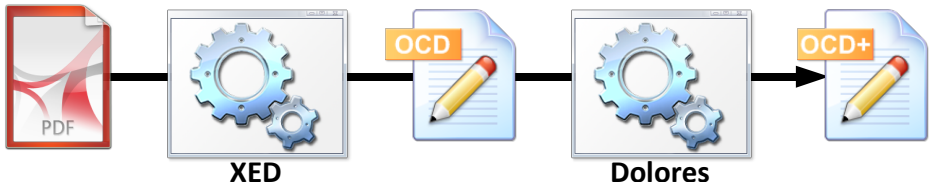


Figure 4.15: The PDF reverse engineering workflow is divided into two main steps handled by two tools called XED and Dolores.

recomposition, document layout analysis and physical structure reconstruction. These steps are described in detail in the Chapters 5 and 6.

Dolores - Document Logical Restructuring - is a tool that aims at recovering the logical structure from PDF documents, based on the results of XED. Indeed, the OCD file format, generated by XED, is used as a complete, standalone, structured, and compact document format. Dolores uses OCD as a pivot format in order to recover the logical structure information from documents. The logical structure information recovered with Dolores is then reinjected into OCD files, now called OCD+. Dolores, presented in detail in Chapter 7, is an interactive and incremental learning system for the automatic generation of document models. Such a learning system is essential to generate logical restructuring models customized to specific document classes and adapted to targeted applications.

5

XED, a Dynamic Tool for the Physical Restructuring of PDFs

Contents

This chapter presents the different steps leading to the physical restructuring of PDF documents. The entire restructuring process is handled by a tool called XED and the result is stored in a dedicated structure called a canonical document. First, Section 5.1 presents the notion of canonical document, a formalism we developped in order to express the content, layout, and structures of electronic documents. Then, Section 5.2 details the implementation of XED, our restructuring tool which is decomposed in three parts: PDF parsing, virtual document generation, and canonical document extraction. Section 5.3 describes the innovative hybrid physical restructuring methodology used by XED. Some evaluations are then presented in Section 5.4. Finally, Section 5.5 highlights the novelty and the contributions of our approach compared to standard ones.

5.1 Canonical Document Definition

A *canonical document* - CD - is an abstract document keeping a clean and highly structured internal representation. It is a formalism that expresses the content, layout, and structures of electronic documents. A CD must ensure the following five properties:

- Universality: a CD must be able to represent any kind of printable document (i.e., documents without interactive elements or time-dependent medias) containing either textual or graphical content.
- Completeness: no loss of visual information should occur during the conversion from a printed document to a CD.
- Uniqueness: text, graphics, and images will be represented in a unique and non-ambiguous manner.
- Homogeneity: textual content has to be organized into homogeneous text blocks containing text lines themselves decomposed in tokens.
- Simplicity: a CD must be as simple as possible in order to facilitate further manipulations; all information should be easily reachable and described thanks to precise, concise, and simple constructs.

In practical terms, we defined a concrete scheme to represent a canonical document: it consists of a preliminary section describing resources and a main section holding the content of one or several pages. Document resources contain both fonts and clips which can be referenced by an ID number. A font is handled as a set of glyphs (vector paths), with their attributes, mapping unicode characters. Clips are represented as vector paths delimiting boundaries of regions in which painting operations can be applied. The canonical document content is represented with three different graphical objects:

- Images, used to reproduce any kind of digital image, are represented as raster graphics.
- Graphics, used to reproduce vector graphics, are represented as geometrical primitives such as lines and curves composing paths.
- Text blocks, used to reproduce textual content, are represented as homogeneous blocks of text. Homogeneity criteria are basically similar font sizes and text spanning over regularly spaced text lines. Text blocks are decomposed into text lines, tokens, and characters.

A canonical document uses only a minimal set of graphical primitives and attributes in order to ensure that two documents that are visually equivalent will produce the same canonical document. Thus, there is only one way to represent a specific kind of data, e.g., fonts description are expressed thanks to a single intuitive mechanism.

5.2 Canonical Document Extraction

The canonical document extraction from electronic documents is handled by a tool called XED - eXtracting Electronic Document. XED is a solution we developped, a PDF extraction

tool able to read and analyze the content of PDF documents. It is a sophisticated system allowing electronic documents to be physically restuctured and stored in a canonical document format. The restructuring process of XED is divided into three main steps: PDF objects parsing, virtual document construction, and canonical document reconstruction. These sequential steps, necessary to convert a PDF document into a canonical one, are illustrated on Figure 5.1.

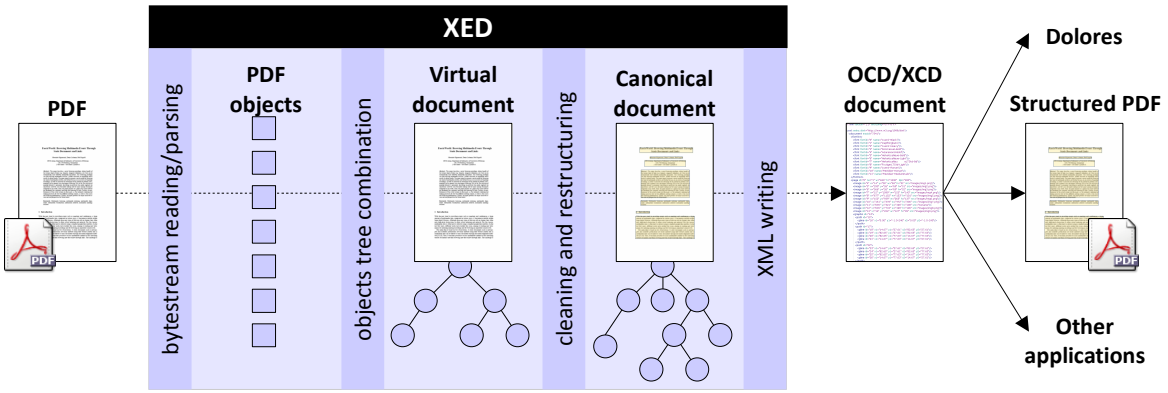


Figure 5.1: The conversion from a PDF file to an OCD one is handled in three distincts steps: the PDF reading and parsing, the creation of the virtual document, and the canonical document extraction.

In the first step, XED parses the source file containing the PDF document and generates a flat list of PDF objects (see Subsection 5.2.1). The second step interprets and combines together these single PDF objects in order to produce new high-level object structures, called *virtual objects*, composing a *virtual tree structure* representing the *virtual document* (see Subsection 5.2.2). This virtual document is an internal interpreted (or rendered) version of the PDF document.

In the third stage, the graphical primitives composing the document, i.e., text, graphics, and images, are analyzed and the physical structure is extracted. During this phase, XED rebuilds a new tree, describing document's structure, content, and layout (see Subsection 5.2). The resulting tree is composed of canonical document objects and represents the canonical document. The physical restructuring algorithm leading to the canonical document is an elaborated hybrid top-down/bottom-up approach which is presented in a dedicated section (see Section 5.3). Finally, a canonical document can eventually be stored in an XML file, either XCD, or OCD, a strong optimized version of XCD (see Chapter 6).

5.2.1 Parsing PDF Objects

Parsing consists in reading a PDF file in order to convert its stream of bytes into a set of PDF objects, i.e., boolean, number, string, name, array, dictionary, stream, and the null object.

These objects are further used for representing every resources and primitives composing the document. Figure 5.2 shows the results of the parsing of the "tiny PDF drawing" file in XED Inspector (our debbuging interface). PDF objects are listed on the left while the content of the current selected object is shown on the right. Objects that are not embedded in composite objects (they are listed on the left of the figure) are called root objects and are referenced both by an identification and a generation number.

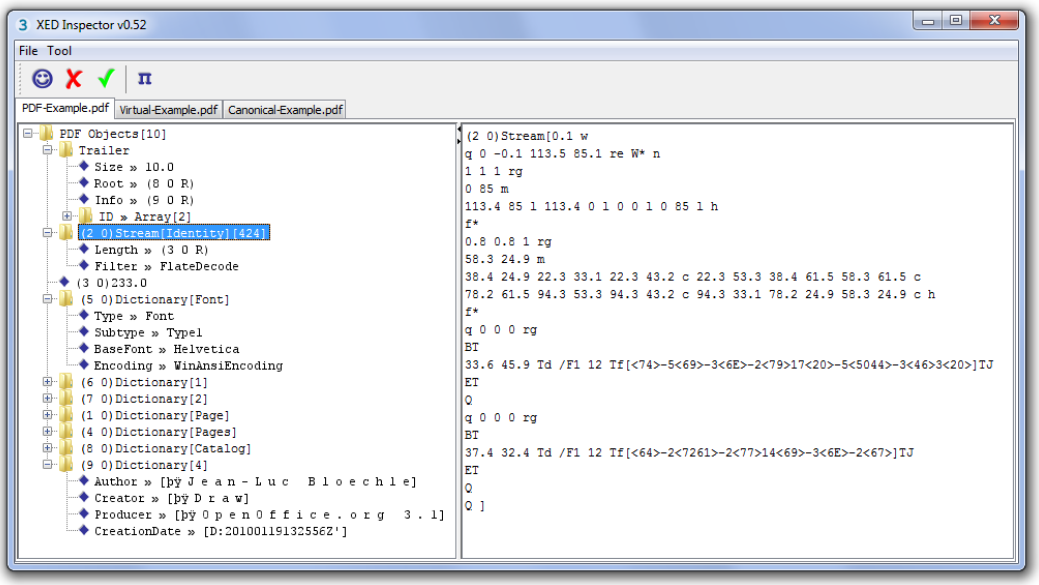


Figure 5.2: PDF objects revealed by the XED Inspector user interface. On the left, the PDF objects are represented as a tree structure, on the right, the content of the selected PDF object is displayed.

The parsing of PDF objects is achieved thanks to two complementary processes: a lexical analysis is done over the raw data, followed by a syntactic analysis that produces a list of uninterpreted PDF objects. Actually, the lexical analyzer (or lexer) and the syntactic analyzer (or parser) work simultaneously. The lexer scans the PDF stream at byte level in order to reconstitute raw PDF tokens. Once a PDF token is reconstituted, it is instantly used to feed the parser in order to progressively build the PDF objects.

Special care has to be taken when dealing with PDF stream objects. Indeed, such objects embed byte streams that also have to be correctly parsed. Stream objects contain a special attribute specifying their size and allowing PDF readers to directly jump to their end. Unfortunately, due to poor PDF writers, content stream sizes may sometimes be inexact or even missing. In such cases, a Boyer-Moore algorithm [15] is used in replacement of the default stream size jumping technique. This algorithm allows the lexer to efficiently parse the content stream by looking for its end pattern, i.e., "end stream". Let us note that this technique is

not totally error prone; indeed, the end pattern may not be univoque since byte streams can potentially contain any sequence of byte values (and thus the end pattern itself).

Since PDF 1.5, root PDF objects themselves may be embedded and encoded in PDF streams. Such a technique was especially developped for the internet accessibility, it is part of the *PDF Linearization* features and allows PDF files to reduce their storage size by compressing PDF streams embedding root objects. Thus, up-to-date parsers also have to process streams containing root objects by first decompressing them and, then, adding their objects to the flat set of root objects.

During the parsing phase, the trailers are located, their contents are used to create a single dedicated trailer object which is the entry point of any PDF document. A PDF file may have many trailers, since each PDF incremental update adds a corresponding updated or complementary trailer. Furthermore, a trailer may be expressed directly thanks to a dedicated structure called an *xref* table, but it can also be embedded in a PDF object.

The xref table also specifies the byte offset of each root object, allowing PDF readers to use a random access to them. However, our parser does not use such features; XED parses sequentially the entire PDF document content, thus avoiding portential parsing errors induced by wrong byte offsets stored in the xref table.

5.2.2 Virtual Document Generation

Once the parsing is done, PDF objects are interpreted and combined together, in order to form high-level object structures, called *virtual objects*, composing the document: page objects, font objects, color space objects, filter objects, etc. The resulting internal PDF document is expressed as a tree of high-level virtual objects and is called the *virtual document*.

The virtual document generation starts by interpreting the PDF trailer content in order to get the reference to a virtual object called *catalog*. The catalog object itself refers to another virtual object containing a tree structure pointing to virtual page objects. The virtual document is then recomposed by interpreting each page object, referenced by the page tree structure, and by interpreting (or rendering) their page content stream. The tree is traversed using a preorder visiting algorithm, the page ordering is reflected by the order in which the tree leaves, i.e., page objects, are visited.

A content stream is most of the time compressed and sometimes encrypted. In this case, it is first decoded in order to access its original ASCII representation. Each content stream is then interpreted and the resulting graphical content of each page is expressed as an ordered list of virtual graphical objects. Each graphical object in a content stream inherits its properties from the current graphics state: transformation matrices, colors, strokes, fonts, etc. Textual elements have their own transformation matrix that are also relative to the current transform matrix. In order to simplify further use, matrices are combined together, so virtual text objects (as well as graphics and images) are expressed thanks to single transform matrices in an absolute coordinate system.

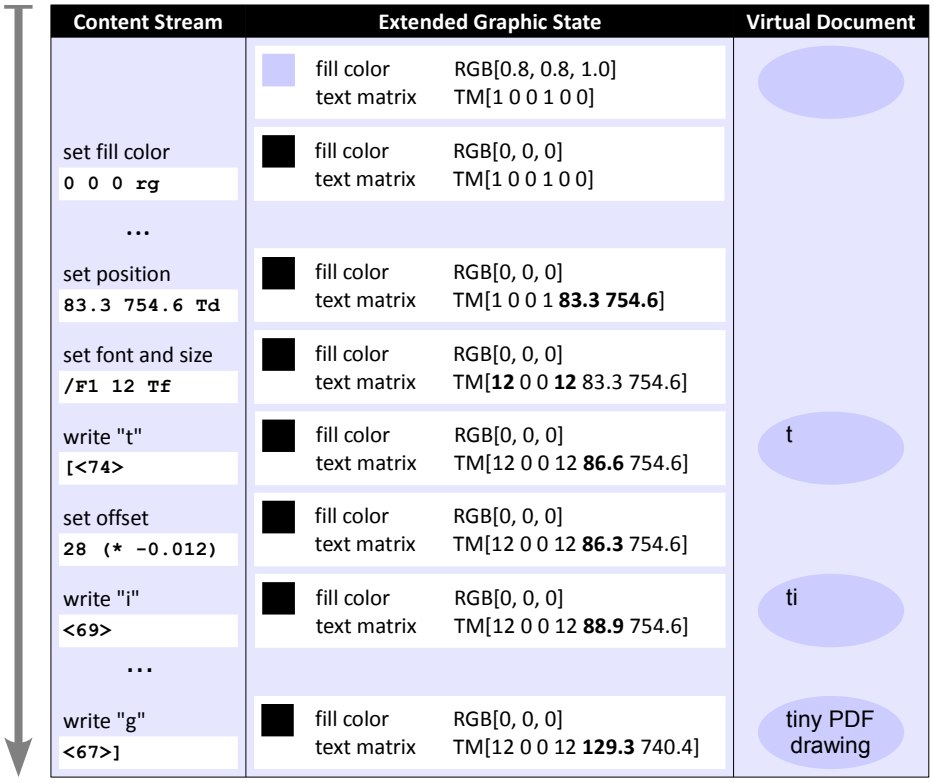


Figure 5.3: Graphical primitives rendering during the creation of the virtual document. The graphics state controls the entire rendering process of a PDF page.

Figure 5.3 illustrates this process: the PDF operators and operands of the content stream of Figure 4.3 are interpreted in order to create the virtual text objects in the virtual document. The same mechanism is used for the two other types of graphical primitives, i.e., graphics and images, so as to generate virtual paths and images.

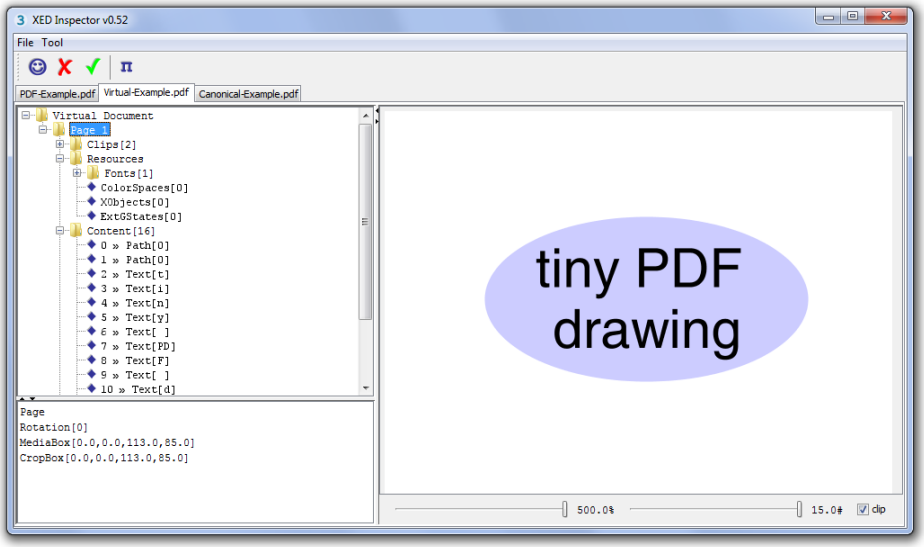


Figure 5.4: Virtual objects revealed by the XED Inspector user interface. On the left, the virtual document is represented as a tree structure, on the right, the selected object is rendered on the screen.

Once all the content streams of all the pages have been interpreted, the virtual document is complete and represented as a tree structure. Figure 5.4 precisely shows the result of the "tiny PDF drawing" example extracted by XED and expressed as a virtual document in our XED Inspector tool. A virtual document node represents the tree root, itself containing pages as child nodes. Each page is in turn composed of virtual graphic objects, i.e., texts, images, and graphics, which correspond to the leaves. The virtual document tree will later be used as the starting point for our physical restructuring process, i.e., the canonical document extraction.

5.2.3 Canonical Document Reconstruction

The canonical document reconstruction is entirely based on the virtual document generated in the previous step. The reconstruction is divided into two main phases: the first one aims at normalizing the virtual document by using a unique representation for each type of virtual object and the second at restructuring the textual content in order to recover the physical structure. The first phase is precisely the subject of this subsection whereas the second one is handled by a hybrid restructuring algorithm that is described in detail in Section 5.3.

Since a virtual document is an abstract and interpreted representation of a PDF file, its internal graphical primitives reflect directly the PDF file content and structure. However, PDF uses many different formats for representing its primitives while a CD supports only one representation for each primitive. Thus, during the canonical document extraction, each virtual primitive is converted to its corresponding univocal canonical primitive.

For instance, in PDF, text can be described by ASCII characters or by means of hexadecimal values. On the opposite, during the CD conversion, the PDF text content, i.e., character codes, is normalized using the unicode standard. The different font systems available in the PDF file format are also transformed into a single canonical font representation. That is, each font is represented as a set of unicodes values paired with their glyphs descriptions (using canonical paths). Again, in PDF, an image can be represented thanks to many image formats using various colorspace systems whereas a canonical document always represents images as raster objects using the sRGB colorspace.

Similarly, in PDF, data streams are often encoded or compressed, e.g., ASCII data streams are generally compressed using LZW whereas raster image data streams use DCT, CCITT Group 3 & 4, etc. During the canonical document extraction, each PDF stream is decoded; then, the information is converted back to a flattened and uncompressed byte stream representation.

Concerning the coordinate system, every canonical graphical primitive is simply positioned thanks to absolute x and y coordinates; an affine transform matrix is also specified in the case of scale or rotation transformations. For instance, let "$m_{00}, m_{10}, m_{01}, m_{11}, m_{02}, m_{12}$" be a canonical transform matrix used to transform the source coordinates (x, y) into destination coordinates (x', y'). The transform operation is achieved by simply multiplying the source coordinates (represented as a column vector) by the transform matrix itself:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} m_{00}x + m_{01}y + m_{02} \\ m_{10}x + m_{11}y + m_{12} \\ 1 \end{pmatrix}$$

Compared to PDF, and thus virtual documents, canonical documents use a different and *normalized* coordinate system. Nowadays, most document formats tend to use the upper-left corner as origin. Thus, whereas PDF uses a y-coordinate that increases bottom-up, our CD format uses a y-coordinate that increases top-down. Such a coordinate system has also the advantage to follow the latin people reading order.

PDF defines a drawing canvas and a page view thanks to both a *MediaBox* and a *CropBox* whose origins may be anywhere in the euclidean space. These boxes define the boundaries of the physical medium and the regions on which the pages are intended to be displayed. On the opposite, a canonical document only defines a page width and height, the origin of the page corresponding to the origin of the coordinate system. Therefore, the transition between a virtual document and a canonical document needs a complete update of the graphical

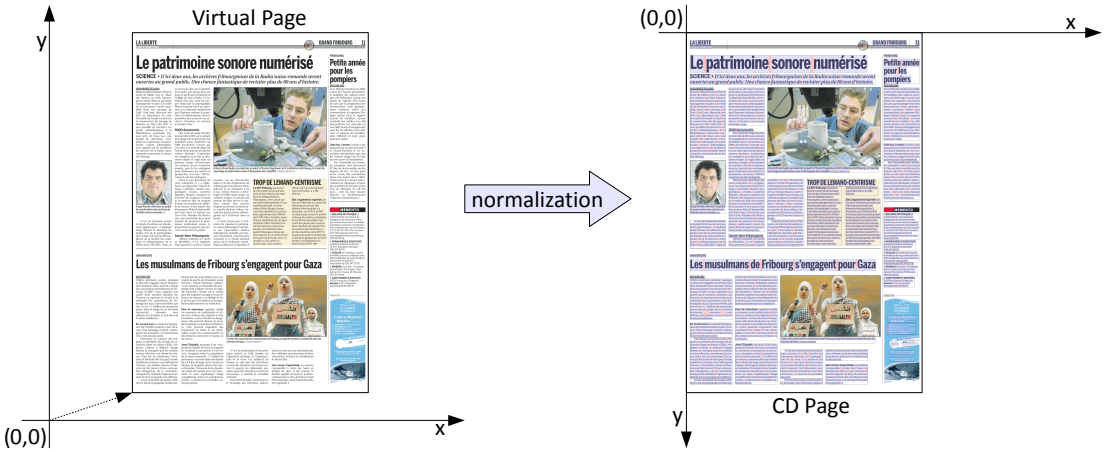


Figure 5.5: An example showing the coordinate system normalization. On the right the standard PDF coordinate system, on the left, the CD coordinate system.

primitive's coordinates and their corresponding transform matrices, as shows Figure 5.5. As well as PDF, a CD uses a customizable distance measure unit expressed in dots per inch - DPI -.

At the end of the canonical document extraction, each virtual object has been transformed into its corresponding canonical object. Each canonical page is therefore composed of virtual images, graphics, and texts objects. Texts objects are themselves composed of blocks, lines, and tokens. The reconstruction of the text structures is an elaborate task that is achieved thanks to an hybrid physical restructuring method which is precisely the subject of the next section, i.e., Section 5.3. Roughly speaking, a text block refers to homogeneous text spanning over one or several text lines that have a regular interline, a similar font size, consistent margins, and include a limited number of font variants.

Figure 5.6 shows the canonical document corresponding to the example "tiny PDF drawing" opened in XEDInspector. The CD is represented as a tree where the root node is the document itself. The root contains two children nodes, the resource node describing the resources used by the document and the pages node containing the single pages.

5.2.4 Achievements

The canonical document extraction is based on the virtual document generation, i.e., the reading of a PDF file. However, several APIs and SDKs providing ways to read and write PDF files already exist. Unfortunately, none of them ensure a complete and detailed access of the entire PDF data, i.e., primitive coordinates, font attributes, glyph shapes, and so on. Therefore, accessing the *deep* content of PDF files implied the development of our own PDF reader. Since PDF is a vast format that incorporates hundreds of different object structures,

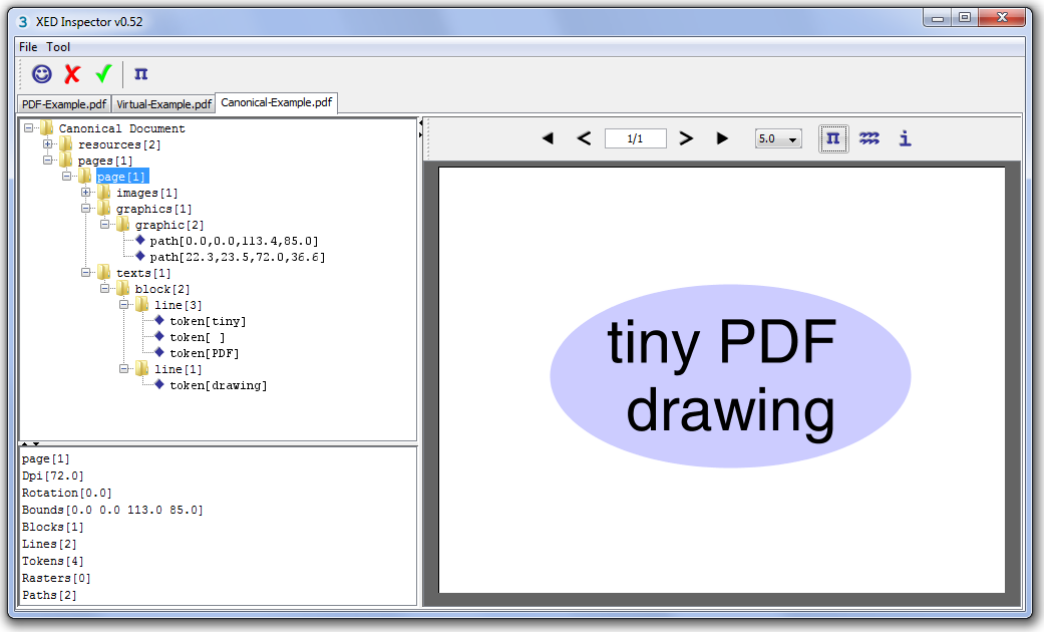


Figure 5.6: The "tiny PDF drawing" canonical document viewed in XEDInspector. The canonical document is represented as a tree structure (on the upper left).

our PDF reader is not yet able to read 100% of the PDF documents, but Adobe Reader itself fails with some old PDF files due to backcompatibility issues.

The genesis of XED goes back to the year 2004, when Maurizio Rigamonti began the implementation of a first PDF reader called Xed [96]. At that time, Xed had no GUI interface and no restructuring capacities; extraction results were direclty stored as SVG files. Although XED is based on Xed (note the typographical difference between "XED" and "Xed"), its implementation has been reviewed from scratch and facilitated by the development of XED Inspector, a complete debbuging environment displaying the results of the various processing steps: PDF parsing, virtual document generation, and canonical document extraction. At each step, XED Inspector displays a standalone pane itself subdivided into two parts: a first pane containing a tree structure representing the processing results is displayed on the left, whereas the corresponding graphical output is drawn on the right. The user can interact with the tree structure to access the internal information of its nodes and leaves, i.e., PDF objects, virtual objects, or canonical objects; the graphical output is instantly updated in consequence.

Let us note that some interesting issues about the virtual document generation have not been presented, such as PDF font systems, raster images, and color profiles decoding. These objects are complex structures demanding strong engineering skills together with a specific knowledge about their internal encoding; this dissertation is not the appropriate place to

describe such intricate engineering details. Still, XED is able to deal with most of the PDF font systems, image formats, and color profiles. For instance, Figure 5.7 shows a truetype font object which is part of a virtual document: font glyphs are printed on the right of the tabbed pane whereas useful font information are available at the bottom left of the tabbed pane.

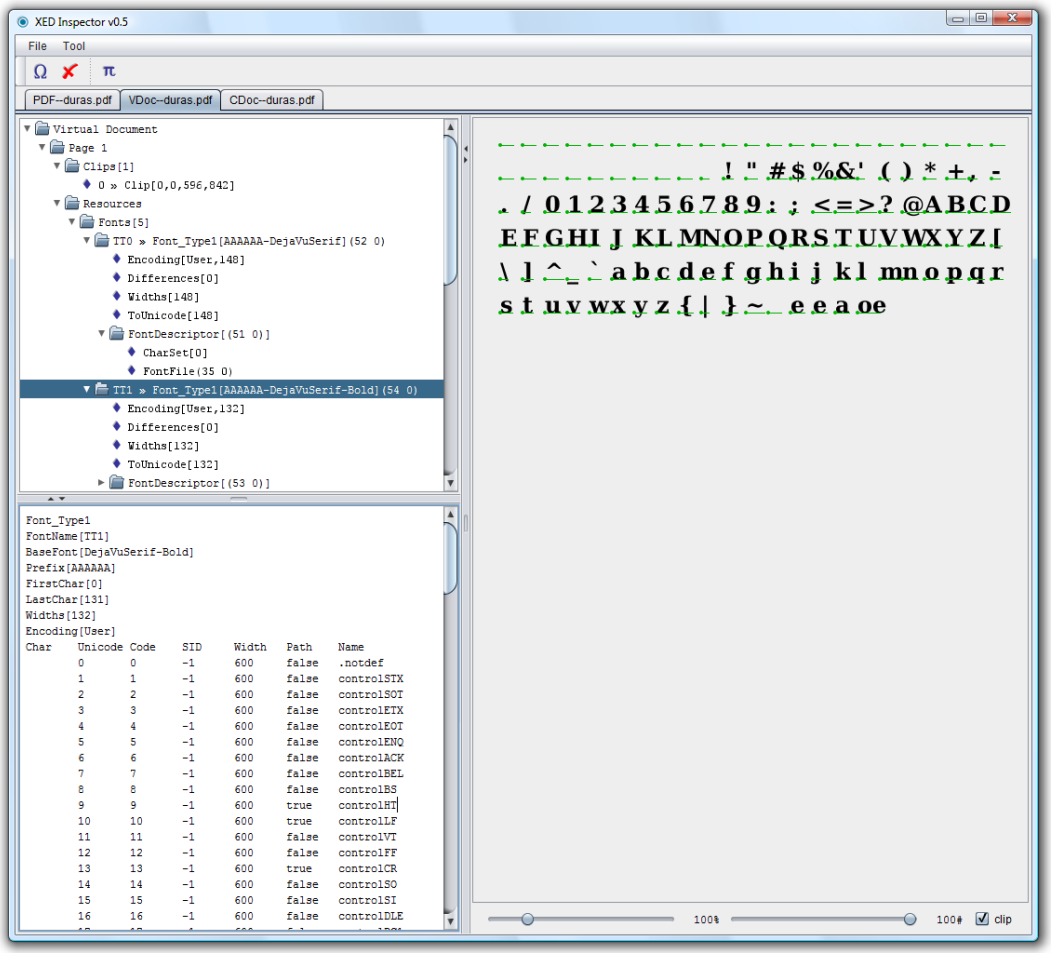


Figure 5.7: A truetype font object displayed by the XED Inspector user interface. XED Inspector is really useful as debugging environment as it provides a complete interface to navigate through PDF, virtual as well as canonical objects, and as it displays their properties.

Many versions of XED and XEDInspector came out these past years. The first processing step of XED, composed of a lexer and a parser, seems to be complete and stable since all the PDF files processed during our research have successfully been parsed. The second processing step of XED, i.e., the virtual document generation, is still error-prone. Nevertheless, it works perfectly with the major part of PDF files. The third processing step of XED, i.e., the canonical document extraction, is completly stable as long as the virtual document generation

has succeeded. Finally, concerning the physical restructuring algorithm itself, it is precisely the subject of the following section.

5.3 A Hybrid Physical Structure Analysis

Since the text segmentation of PDF documents is totally unpredictable, software relying on PDF content cannot make any assumption about their textual segmentation and ordering. Consequently, when analysing the physical structure of PDF textual content, one can only rely on the single characters, i.e., the invariant and smaller textual entities.

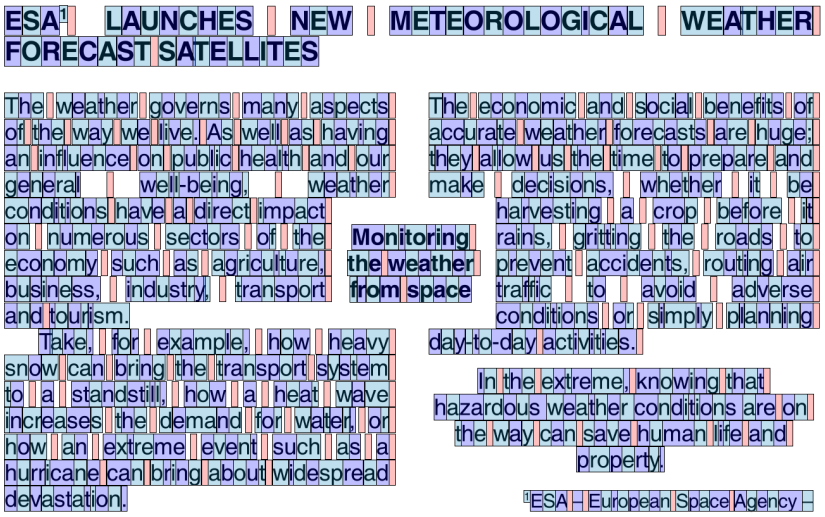


Figure 5.8: The "ESA" example showing the raw segmentation of a representative PDF file. Text primitives are emphasized by alternating brighter and darker blue rectangles.

Figure 5.8 shows, for instance, a concrete example of PDF text segmentation; the original file has been created with OpenOffice Writer and exported as PDF. This example is very representative and therefore will be used further in this section in order to illustrate the result of each physical restructuring step. From now on, we will call it the "ESA" example ("ESA" is an acronym standing for "European Space Agency").

As follows, this section displays in details our hybrid physical structure analysis algorithm. The first subsection gives an overview of the restructuring algorithm. Then, we present the dynamic thresholding, a key concept of our algorithm. The following three subsections describe the algorithm itself, i.e., text preprocessing, bottom-up phase, and top-down phase. Finally, the last subsection exposes the development life cycle of the algorithm.

5.3.1 Algorithm Overview

The analysis of the physical structure is handled by an hybrid algorithm which has both bottom-up and top-down phases. Our restructuring algorithm uses only a restricted set of textual primitive properties, i.e., unicode value, font size, character width, position, and font name. This choice has been made for two main reasons. First, PDF does not ensure that white space, interline, and other typographical operators are correctly used, or even used at all. Second, our physical restructuring system is also fully functional when applied on electronic document coming from scanned and OCR-ized hard copies. Indeed, most of the time, such systems provide only the basic and essential glyph properties. Therefore, our algorithm is designed to restructure both electronic documents as well as OCR-ized paper documents by using only electronical textual primitives.

Actually, it proceeds according to three main phases, themselves decomposed into steps leading to a clean and canonical representation.

1. Text preprocessing
    - *text layering* creates a layer for each existing text orientation, restructuring is then applied for each layer sequentially.
    - *text homogenization* first removes all white spaces from the text content, then, it splits the text primitives regarding the character types, i.e., number, punctuation, symbol, letter.
    - *text reordering* simply sorts each text primitive according to its x-coordinate.
2. Bottom-up phase
    - *text tokenization* merges the text primitives into tokens, i.e, words, numbers, symbols, and punctuation marks, using a dynamic threshold.
    - *text linearization* merges the tokens into text lines using a dynamic threshold.
    - *text clustering* merges the text lines into text blocks using a dynamic threshold.
    - *retroactive merging* recovers the over-segmented text lines from each text block.
    - *post-linearization* recovers the over-segmented text lines and text blocks induced by text justification.
3. Top-down phase
    - *interline change detection* splits text blocks by precisely detecting interline changes.
    - *text phagocytosis* recovers small text elements such as subscripts and superscripts in order to intergrate them in their overlaying text block.
    - *white space generation* simply adds white spaces between the tokens of a line.
    - *item detection* splits text blocks by looking for item patterns.

- *alignment change detection* splits text blocks by comparing the consecutive text lines left and right alignments.

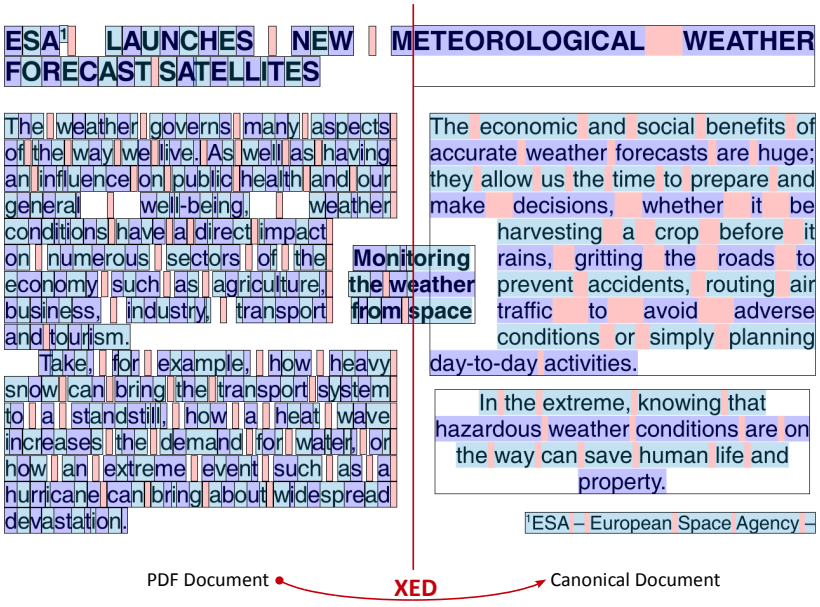


Figure 5.9: The "ESA" example showing the raw segmentation of a PDF file (on the left) together with its canonical segmentation (on the right).

These steps are performed on the original PDF segmentation and result in the canonical text segmentation (see Figure 5.9). Bottom-up and top-down phases use text operations such as merging and splitting. Such operations require the definition of threshold values in order to decide whether or not we do apply a given operation.

5.3.2 Dynamic Thresholding

A relevant feature of our system is its ability to generate a canonical document from any PDF file without any particular customization. Indeed, the physical structure extraction is performed without specific tuning, because all the thresholds are generated dynamically, using ratios of text extracted features; e.g., word spacing, leading, horizontal/vertical distance are all normalized by the font size.

At this stage, we present some key formulae used by our physical restructuring algorithm. Let i and j be two text primitives, $fontsize$ a function returning the font size of a text primitive, and *baseline* a function returning the y-coordinate of a text primitive. We define the *normalization factor* between two text primitives as follows:

$$\nu_{ij} = minimum(fontsize(i), fontsize(j)) \tag{5.1}$$

We now define a dynamic precision threshold Θ_{ij} between two text primitives i and j. This precision threshold is a normalized and dynamic version of the $\sigma_{precision}$ ratio (because ν_{ij} represents the minimum font size between i and j); it is expressed as presented bellow:

$$\Theta_{ij} = \sigma_{precision} \cdot \nu_{ij} \tag{5.2}$$

It is now possible to use the dynamic and normalized precision threshold to compare font sizes between text primitives. That is, two text primitives i and j have similar font sizes if the following condition is verified:

$$|fontsize(i) - fontsize(j)| < \Theta_{ij} \tag{5.3}$$

The same reasoning is applied to compare the baselines between text primitives. That is, two text primitives i and j have similar baselines if the following condition is verified:

$$|baseline(i) - baseline(j)| < \Theta_{ij} \tag{5.4}$$

The physical structure extraction algorithm described in details in the next subsections greatly benefits of the introduced concepts, i.e., similar font sizes 5.3, similar baselines 5.4, and precision threshold 5.2.

The physical restructuring steps use other ratios that are also represented with the "σ" symbol. As presented above, the $\sigma_{precision}$ ratio is used to determine whether or not two values are similar. The $\sigma_{tokenization}$, $\sigma_{linearization}$, and $\sigma_{clustering}$ ratios are used to determine if the distance between two text primitives corresponds to a token, a text line, and a text block, respectively. These three ratios are therefore used in conjunction with the font size (dynamic thresholds) in the tokenization, linearization, and clustering steps.

All these ratios have been acquired experimentally and refined iteratively over an heterogeneous PDF corpus including e-books, newspapers, scientific papers, journals, and various OCR-ized documents. The chosen ratios (thus thresholds) tend to be minimal, over-segmentation being preferable than under-segmentation when dealing with textual content.

Name	Value
$\sigma_{precision}$	0.25
$\sigma_{tokenization}$	0.08
$\sigma_{linearization}$	0.80
$\sigma_{clustering}$	0.80

Table 5.1: Physical structure extraction ratios. Such ratios are used to generate the dynamic thresholds according to the document's font sizes.

Table 5.1 presents the ratio values, these are the ones that best segment the pre-cited documents; trade-offs always led to minimal ratios, thus favouring over-segmentation as stated earlier.

5.3.3 Text Preprocessing

Electronic documents as well as paper documents may contain rotated textual content. Advanced physical restructuring techniques should take the text orientation into account in order to merge and split homogeneous text primitives in all directions. Moreover, in PDF documents, text primitives often contain superfluous and inadequate white spaces that have to be removed. Text primitives also need to be reordered before further processing. Hence, our preprocessing phase is divided into three steps: text layering, text homogenization, and text reordering.

Text Layering

The first text preprocessing step is called *layering*. Indeed, a text layer α is created for each existing angle used for drawing text primitives (angles are expressed counterclockwise). As documents often contain a unique zero degree layer, this step is most of the time unuseful. However, for each non-horizontal layer, text primitives are rotated back to an angle of 0 degree, i.e., a rotation of $-\alpha$ about the origin of the coordinate system is performed. All the text primitives have then an horizontal orientation and each of them is assigned to its α-layer.

All the following physical restructuring steps are applied to each α-layer sequentially, one at a time. This layering step is an elegant way to avoid further complex sine and cosine transformations. In this manner, text primitives are now horizontally oriented, still they are grouped together thanks to α-layers. Each α-layer can now be processed individually and, since their text primitives are horizontal, the computation of their bounding boxes overlapping and merging is trivial.

Text Homogenization

The second text preprocessing step, called *homogenization*, allows the text primitives to be homogenized; this step is required since the original segmentation of the PDF text primitives is not homogeneous.

Each text primitive is trimmed in order to remove its potential superfluous white spaces. Standalone white spaces are simply removed, whereas white spaces between characters of an under-segmented text primitive are used to split it. This step allows to remove in particular meaningless white spaces overlapped by consecutive letters of a single word.

Text primitives are also split before and after numbers, punctuation marks, and special characters. This results in a text segmentation that contains only primitives having homogeneous character sequences, i.e., numbers, punctuations marks, symbols, and letter sequences (see Figure 5.10).

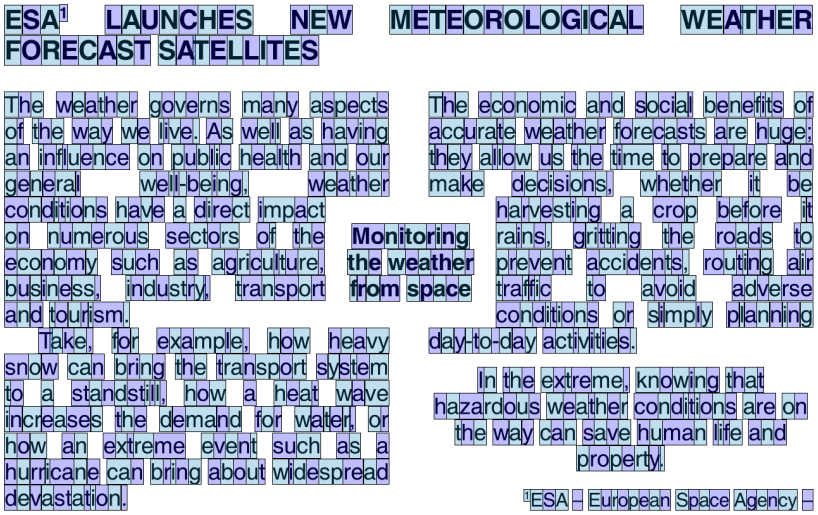


Figure 5.10: The result of the homogenization step on the "ESA" example. The white spaces have been removed and the text primitives have been split before and after numbers, punctuation marks, and special characters.

Text Reordering

The text *reordering* step rearranges all text primitives according to their x- and y-coordinates. Text primitives are each compared to the others until reordering is completed. The algorithm used is a slightly optimized merge sort that is fast and stable, i.e., it is guaranteed to run in $n \cdot log(n)$ and runs substantially faster on nearly sorted lists; moreover it is stable, i.e., it doesn't reorder equal elements.

This method ensures that text primitives are sorted horizontally or vertically before the merging steps occur. Although text reordering is described as a preprocessing step, the reordering is performed each time it is needed. For instance, an x-coordinate sort is performed before the tokenization and linearization steps while an x-coordinate sort followed by a y-coordinate sort is performed at the end of the restructuring.

5.3.4 Bottom-up Phase

Once the preprocessing done, our algorithm proceeds bottom-up in order to merge over-segmented text primitives together. The following steps reconstruct the syntactical units (tokens), group them into text lines, themselves grouped into text blocks.

Text Tokenization

The *tokenization* step merges the text primitives in order to correct over-segmented syntactical units. Text primitives are merged together only if their horizontal distance is less than

a dynamic threshold (Equation 5.5), according to their homogeneity: similar font sizes and baselines (see Equations 5.3 and 5.4), identical colors, and identical sequence types, i.e., digit, punctuation, symbol, character.

Figure 5.11: The result of the tokenization step on the "ESA" example. Text primitives have been merged into tokens given a dynamic horizontal threshold.

Let i and j be two x-ordered text primitives, x_{min}, and x_{max} two functions computing the minimum and maximum x-coordinates of text primitives. Given $\sigma_{tokenization}$ (see Table 5.1) and ν_{ij} (Equation 5.1), text primitives merging occurs only if the following condition is satisfied:

$$\Big((x_{min}(j) - x_{min}(i)) > 0\Big) \wedge \Big((x_{min}(j) - x_{max}(i)) < \sigma_{tokenization} \cdot \nu_{ij}\Big) \tag{5.5}$$

This step results in isolated syntactical units called *tokens* such as words, numbers, punctuation marks, and special characters. These new-segmented entities correspond to the canonical document text primitives (see Figure 5.11).

Text Linearization

The *linearization* step merges the tokens into text lines. Tokens are merged together only if their horizontal distance is less than a dynamic threshold, (Equation 5.6) according to their homogeneity: similar font sizes and baselines (see Equations 5.3 and 5.4).

Let i and j be two x-ordered tokens, and x_{min}, x_{max} two functions computing the minimum and maximum x-coordinates of tokens. Given $\sigma_{linearization}$ (see Table 5.1) and ν_{ij} (Equation 5.1), tokens merging occurs only if the following condition is satisfied:

$$\Big((x_{min}(j) - x_{min}(i)) > 0\Big) \wedge \Big((x_{min}(j) - x_{max}(i)) < \sigma_{linearization} \cdot \nu_{ij}\Big) \tag{5.6}$$

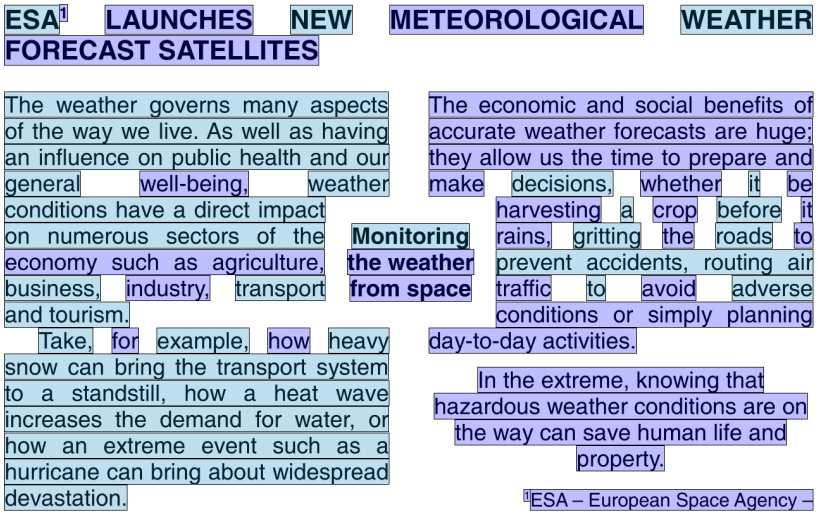


Figure 5.12: The result of the linearization step on the "ESA" example. Tokens have been merged into lines given a dynamic horizontal threshold.

The result of this step is a text segmentation that partially reconstitutes text lines (see Figure 5.12). The threshold ratio "$\sigma_{linearization}$" is intentionally low, indeed, the distance between two tokens of a same line is sometimes bigger than the distance between two columns. Thus a low "$\sigma_{linearization}$" avoids tokens belonging to two adjacent text columns to be merged.

Text Clustering

During the *clustering* step, text lines are merged vertically into blocks by applying a region growing algorithm.

1. A new cluster is created by removing the first text line from the non-clustered line set.
2. Each non-clustered text line is successively tested against every line of the current cluster. A merging occurs only if the vertical distance between a non-clustered text line and a current cluster line is less than a dynamic threshold (Equation 5.7), the two text lines having to possess similar font sizes (Equation 5.3).
   - When a text line is merged into a cluster, it is first added to its clustered lines. Then, the non-clustered text line set is updated by removing the clusterized text line. At this point, the clustering process restarts at step 2.
   - Once the current cluster does not grow anymore, i.e., the non-clustered text line set is stable until the end of step 2, the clustering process restarts at step 1.
3. Once all the text lines belong to a cluster, the clustering process stops.

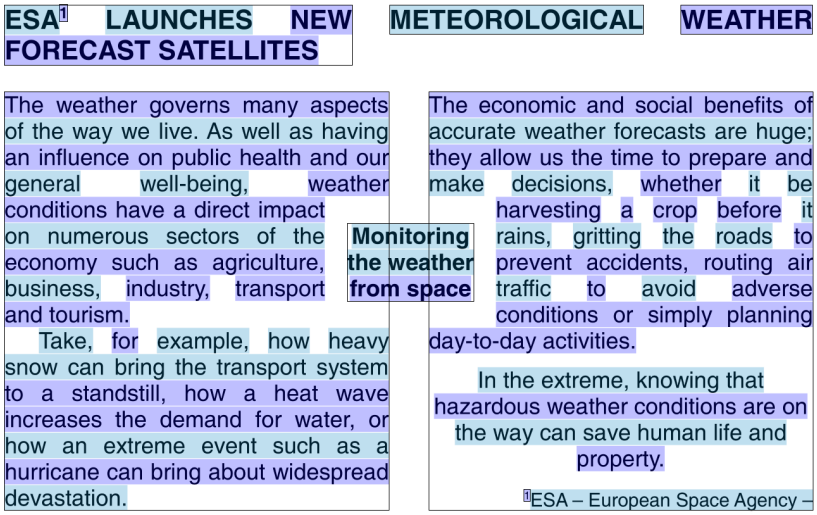


Figure 5.13: The result of the clustering step on the "ESA" example. Lines have been clusterized in text blocks thanks to a dynamic vertical threshold.

Let i and j be two text lines, $proj_h$ the horizontal projection of a text line, and $distance$ the mininum distance between two text lines. Given $\sigma_{clusterization}$ (see Table 5.1) and ν_{ij} (Equation 5.1), the clustering occurs only if the following condition is satisfied:

$$\Big(proj_h(i) \cap proj_h(j) \neq \emptyset\Big) \wedge \Big(distance(i,j) < \sigma_{clustering} \cdot \nu_{ij}\Big) \tag{5.7}$$

An important property of our algorithm is the use of non rectangular cluster bounds during the processing: a cluster bound is expressed as the union of its text line bounds. This technique allows overlapping block of texts to be correctly detected by avoiding unwanted line merging. Figure 5.13 shows the result of the clustering process on the "ESA" example. Each text block is surrounded by a thin and solid line whereas text lines are surrounded by thin dashed lines. We observe that the enclosed text block has been correctly detected.

Retroactive Merging

The *retroactive merging* step is applied over each text block in order to recompose over-segmented lines. This over-segmentation can be induced by the low dynamic thresholds generated during the linearization step. This problem often arises in case of justified lines: the distance between strings increases and, consequently, text lines may not be correctly merged.

This retroactive merging step precisely corrects all the over-segmentation errors inside text blocks. Each text block is scanned successively, text line baselines are compared to each others, and text lines having similar baselines (see Equation 5.4) are merged together.

Figure 5.14 shows the "ESA" example after the retroactive merging process. We observe that, except for the title, the text lines are now correctly segmented.

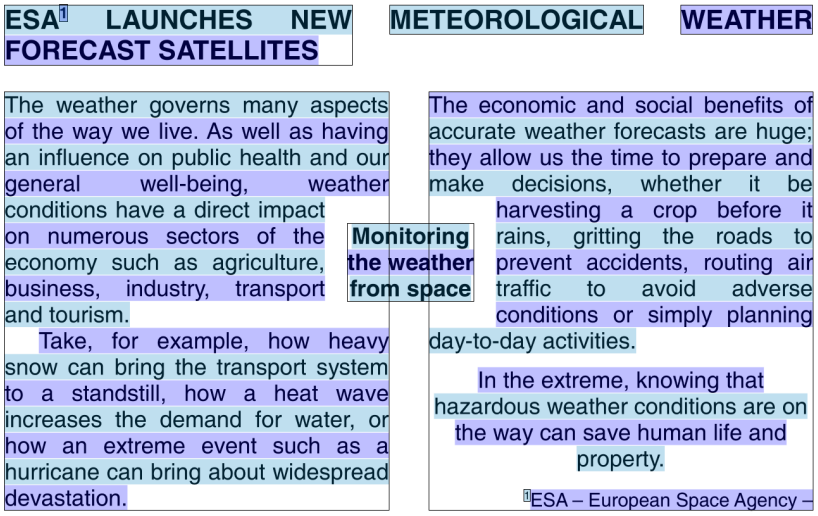


Figure 5.14: The result of the retroactive merging step on the "ESA" example. Over-segmented lines of a same text block have been correctly merged.

5.3.5 Top-down Phase

The top-down phase contains the last steps of the canonical document physical restructuring process. These steps try to split the under-segmented text blocks thanks to the interline changes and text alignments.

Interline Change Detection

The *interline change detection* step detects sudden interline changes inside text blocks in order to split them into homogeneous paragraphs. Each text block is scanned successively, the differences between the consecutive interlines of a text block are computed. If this interline difference is greater than a dynamic threshold relative to the font size, the wider interline is then used to split the text block in two pieces.

More formally, let i, j, and k be three consecutive y-ordered text lines belonging to the same text block:

If $|(baseline(j) - baseline(i)) - (baseline(k) - baseline(j))| > \sigma_{precision} \cdot \nu_{ik}$, then

a) split before j, if $(baseline(j) - baseline(i)) > (baseline(k) - baseline(j))$,

b) split after j, otherwise.

Figure 5.15 shows the result of the interline change detection step on the "ESA" example. A text block splitting occurs before the centered text lines because of the interline change.

Figure 5.15: The result of the interline change detection step on the "ESA" example. Text blocks containing various interlines have been split.

Text Phagocytosis

The *text phagocytosis* step merges the text blocks overlapped by smaller text primitives. Until now, such text primitives have been left out because of their small fontsizes. Indeed, only text primitives having similar fontsizes have been merged. However, latin languages often use superscript and subscript characters inside their text flow. For instance, this document uses a lot of subscript tokens such as $\sigma_{precision}$, ν_{ij}, etc. Superscript tokens are also quite common, e.g., $M^{me}Bovary$, $14^{th}of July$, etc.

This step proceeds as follows: the bounding box of each text block is expanded successively thanks to $\sigma_{linearization}$ and $\sigma_{clustering}$. If the expanded bounding box fully contains one of the remaining text blocks, the font sizes are compared. Therefore, if the font size of the expanded text block is bigger than the font size of the contained text block, and if the difference between the font sizes is bigger than Θ_{ij}, i.e., the equation 5.3 is not satisfied, the text blocks are merged together.

The result of the text phagocytosis step is presented in Figure 5.16. We observe that the two superscript "1" are successfully merged by their overlapping text blocks.

White Space Generation

The *white space generation* step simply adds white spaces between the tokens. Since all white spaces have been removed during the preprocessing phase (see Subsection 5.3.3), they need to be put back between each token of a line, if needed. Thus, nonexistent white spaces are added between consecutive tokens of a line if their distance is greater than the tokenization

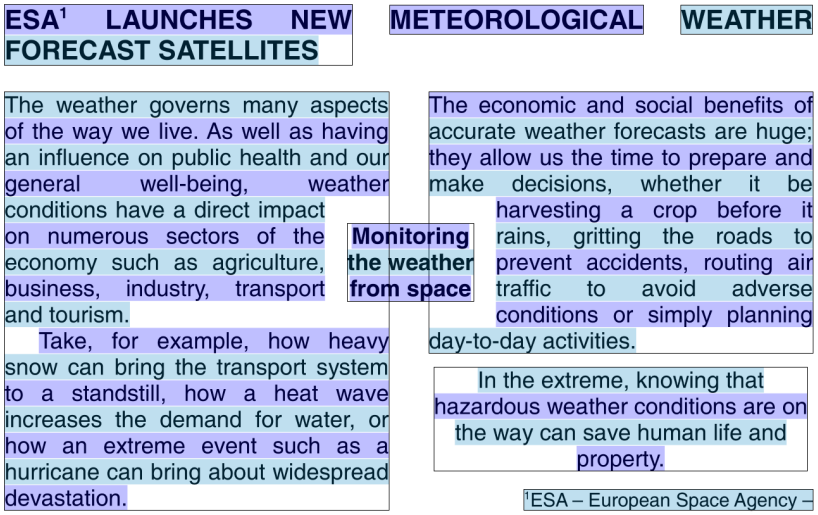


Figure 5.16: The result of the text phagocytosis on the "ESA" example. Small text elements have been embraced by their overlapping text blocks.

distance, i.e., the Equation 5.5 is not satisfied. Figure 5.17 shows the result of the white space generation.

Item Detection

The *item detection* step splits all the text blocks containing a list of items thanks to the following three regular expressions:

1. `\A\[\d+\]\s` detects bibliographical items, such as "[1] G. Flaubert, Madame B., ...".
2. `\A\d+\.\s` detects enumeration items, such as "1. the first item", "2. the second item".
3. `\A\bullet\s` detects bullet items, such as "• an item", "• another item".

These regular expressions are written in a Java formalism, i.e., "`\A`" means "character string beginning", "`\d+`" means "one or more digit(s)", and "`\s`" means "one and only one white space". Other symbols such as "`\[`", "`\]`", "`\.`", and "`\bullet`", simply means "[", "]", ".", and "•", respectively.

Thus, during the item detection step, each text block is once again scanned. A split occurs only if a text line matches one of the three precited regular expressions.

Alignment Change Detection

The *alignment change detection* step compares consecutive text line left and right alignments in order to detect indentation. The purpose is to split under-segmented text blocks into paragraphs thanks to their left and right justifications. For instance, first lines of paragraphs

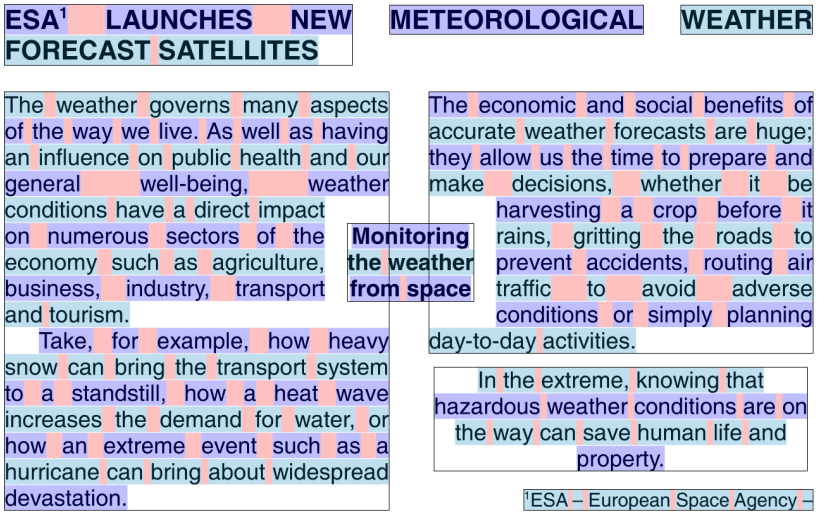


Figure 5.17: The result of the white space generation on the "ESA" example. White spaces have been put back between tokens in order to get the complete textual sequence.

are often indented and, therefore, it is possible to use this information to split an under-segmented text block into several new text blocks, i.e., paragraphs.

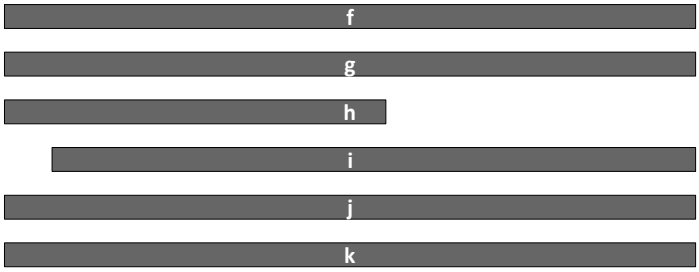


Figure 5.18: This figure presents six consecutive text lines, each of them is referenced by a letter. Lines f, g, h belong to a first paragraph, whereas lines i, j, k belong to a second one.

Two rules have been implemented in order to split the under-segmented text blocks. Let i be the current text line, then h is the previous one and j the next one, and so on (see Figure 5.18):

1. If (g, h) and (h, i) are not right-aligned, and h is indented, and (f, g) and (i, j) are strictly right-aligned, split the text block before i. Two text lines i and j are strictly right aligned if $|x_{min}(i) - x_{min}(j)| < \Theta_{ij}$.

2. If (h, i) and (i, j) are not left-aligned, and i is indented, and (g, h) and (j, k) are strictly left-aligned, split the text block before i. Two text lines i and j are left aligned

if $|x_{max}(i) - x_{max}(j)| < \Theta_{ij}$.

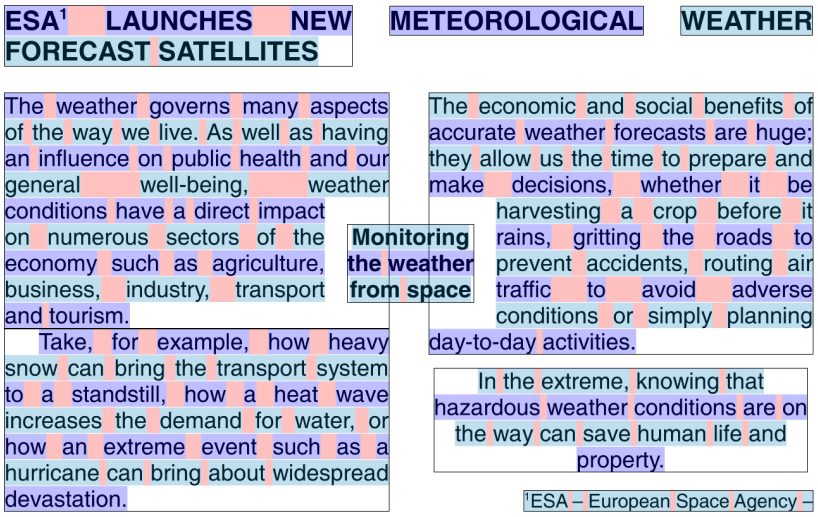


Figure 5.19: The result of the alignment change detection on the "ESA" example. Text blocks have been split again in order to get homogeneous paragraphs.

The first rule works pretty well over fully justified text, whereas the second one works both on left aligned and justified text. Non existing text lines are supposed to be aligned with others, i.e., if f, g, j, and/or k do not exist, the alignment test always gives a positive answer. The text block splitting takes place only if it occurs between a line ending and a line beginning: a line ends/begins if its last/first character is a symbol, an uppercase character, or if its text font changes. Figure 5.19 shows the result of the alignment change detection step, the indented paragraph is efficiently detected.

Post-linearization

The *post-linearization* step tries to recover the last over-segmented text lines thanks to the previously reconstructed text blocks. Indeed, the retroactive merging recovered the text line over-segmentation inside text blocks, unfortunately over-segmentation may also exist between text blocks. For instance, in Figure 5.14, we observe an over-segmentation of the main title which is due to the full justification of the first text line. Since the retroactive merging step recovered the over-segmented text lines into each text block (intra-block merging), it is now possible to extrapolate their white spacing outside them in order to recover the over-segmented lines between adjacent blocks (inter-block merging).

Let i and j be two consecutive x-ordered text lines, a line merging occurs only if the following conditions are satisfied:

1. i or j must have a parent text block containing only one line, i.e., the line itself.

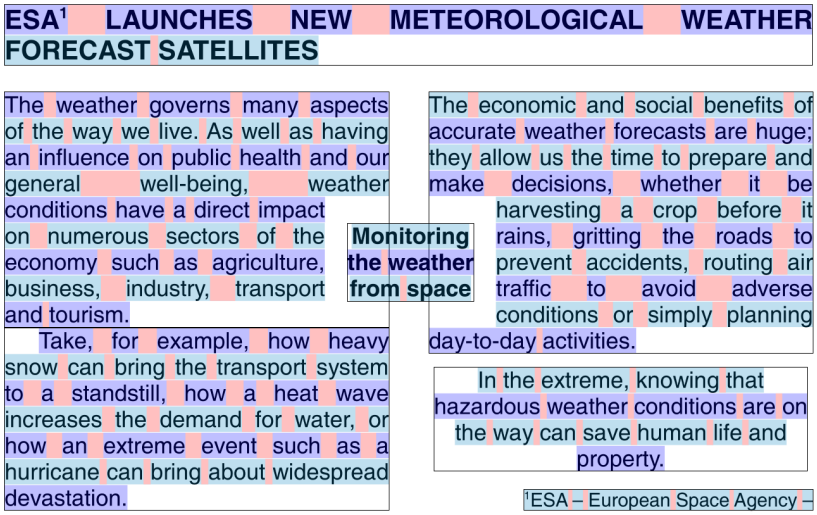


Figure 5.20: The result of the post-linearization step on the "ESA" example. Over-segmented lines belonging to different text blocks have been merged together.

2. i and j must have similar baselines (Equation 5.4).

3. i must contain at least one white space whereas j must not contain any white space, or vice-versa, indeed, we want to merge an existing line with a not yet merged line.

4. i and j must have a distance similar to the white space width of i or j, according to the precision threshold Θ_{ij} (Equation 5.2).

The result of the post-linearization step is shown in Figure 5.20. The title tokens are now entirely merged thanks to the white space width checking. The two text colums have not been merged together thanks to the first and third conditions, ensuring that a clustered text line is only merged with a non-clustered text line.

5.3.6 Algorithm Development Life Cycle

Many directions have been investigated during the development of the physical restructuring algorithm. Only the useful and relevant steps have been kept and presented above.

The preprocessing of text primitives was not part of the preliminary versions of our restructuring process. While inspecting carefully the first segmentation results with XEDInspector, we rapidly came to the conclusion that an efficient physical restructuring process based on PDF was impossible without a cleaning of the raw text primitives.

The retroactive merging step encounters the same impediment since it has been conceived while inspecting text segmentation results with XEDInspector. Hence, we actually noticed that text lines being part of justified paragraphs were often over-segmented. We first tried to

increase the $\sigma_{linearization}$ value, but it resulted in an under-segmentation (adjacent columns were merged together); therefore we determined to add the retroactive merging step.

The top-down phase was totally absent from our first physical restructuring process. Then, we adjucated to use the result of our text block segmentation with concrete tools such as Dolores, our logical restructuring tool (see Chapter 7). We observed that the practical use of our first canonical document restructuring process with complex document classes such as newspapers was inadequate, i.e., text segmentation was clearly too sparse. Therefore, we redesigned our physical restructuring process in order to fill in this gap. As our previous text segmentation heuristics only used a bottom-up scheme, we naturally chose to add a top-down phase in order to split our previous text blocks into new finer ones, i.e., homogeneous paragraphs.

Concerning the paragraph indentation detection step, primarily attempts used the rectangular text block bounds as left and right alignment references. Results were clearly not satisfying with documents having complex structures. For instance, overlapping text block bounds are common in newspapers and in this manner real text line bounds may be quite distant from the rectangular text block bounds. Henceforth, we agreed to detect the relative changes between adjacent text lines left and right alignments.

The post-linearization was the last step to be added to our algorithm. Actually, we remarked that same over-segmentation errors tend to appear; in that account, we inserted this post-linearization step right after the retroactive merging one, in order to benefit from the freshly inferred white spaces.

A great amount of work has been done to find the right steps leading to a successful physical restructuring algorithm. The final version of our algorithm is the result of our long investigations and of practical experiences matured during the successive feedbacks provided by Dolores, our logical structure recovering system (see Chapter 7).

5.4 Evaluation of the Physical Structures Extraction

First of all, our restructuring algorithm is specifically adapted for Latin languages; Arabic and oriental languages are not currently upheld.

Our canonical document performances have been assessed through a qualitative evaluation. Our text blocks segmentation algorithm has been applied on a set of representative documents: e-book, website, and newspaper. We chose to evaluate our algorithm on such documents, because of their disparity. For instance, newspapers have quite complex layouts whereas e-books have a basic one-column presentation. Figure 5.2 shows the percentage of correct text blocks segmentation in respect to human judgment.

More precisely, the PDF content of each document has been extracted, cleaned, and restructured in our canonical document representation. These results have been further visualized in XEDInspector, allowing an expert reader to appreciate the quality of our reengineering

Document title	number of text blocks	number of errors	correctness
E-book - Anna Karenina, Leo Tolstoy	11'581	0	100%
TSR Website - 42 TV Schedules 2005/09/21-27	4'525	42	99.07%
Swiss newspaper - La Liberté 2009/01/16	1'410	16	98.87%
French newspaper - Le Monde 2009/01/16	1'827	35	98.08%

Table 5.2: We evaluated various documents having layouts going from very basic to quite complex. The results of our canonical document segmentation algorithm on these documents are clearly satisfying.

system according to the specification of the canonical document.

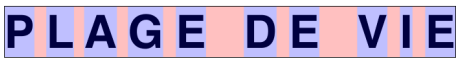


Figure 5.21: An example of wrong token segmentation in the Swiss newspaper "La Liberté".

The attentive reader certainly noticed that token and line segmentations have not been reported in Table 5.2. Indeed, the segmentation into lines is intrinsically related to the segmentation into blocks and therefore, once a text block is correctly segmented, its text lines are correctly segmented too (thanks to the retroactive merging). The same analysis can be applied to the segmentation into tokens, except for some scarce cases where characters of a single word are adjusted using white spaces, thus moving away adjacent characters (see Figure 5.21).

Figure 5.22 and Figure 5.23 show the results of our physical structure extraction algorithm on an e-book (Huckleberry Finn) and a newspaper extract (Le Monde 2009/02/12), respectively. The segmentation is divided into tokens, text lines, and text blocks: tokens are highlighted in light blue, white spaces are highlighted in light red, and text blocks are surrounded by thin rectangles. The image rendering has been completely performed by XEDInspector using uniquely the information stored in our extracted canonical document.

5.5 Contribution of our Approach

The physical structure analysis of electronic document is a great and open issue. Standard document analysis techniques use the raster image as a base in order to extract the physical structure, i.e., bottom-up techniques try to merge image primitives (pixels or connected components) together whereas top-down techniques use the whole image and divide it into homogeneous regions. Few works based on electronic documents have been realised and none of the existing do provide a complete physical restructuring process (see State of the Art, Section 3). Some systems even use the rasterized version of electronic documents in order

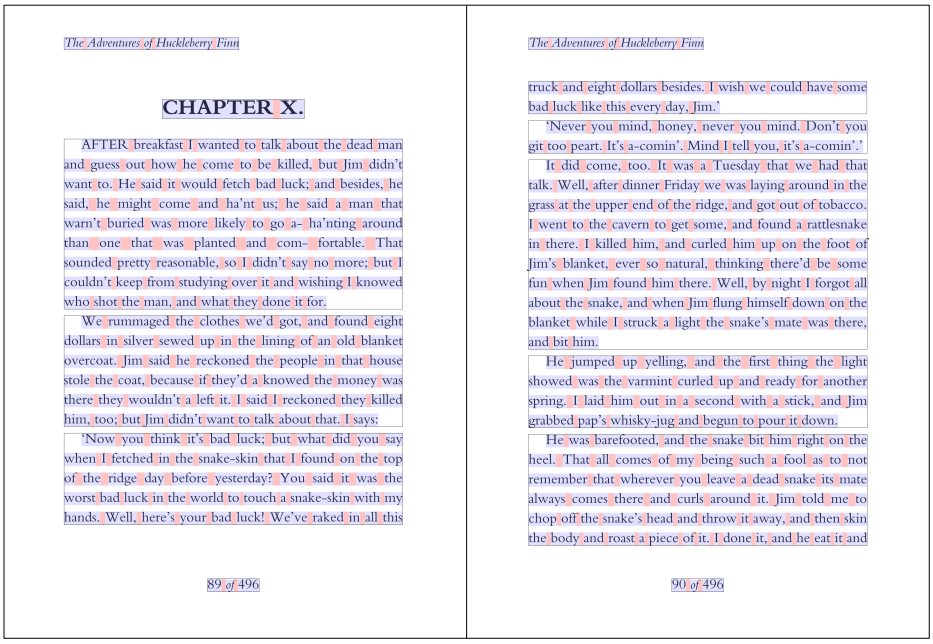


Figure 5.22: An excerpt of the Huckleberry Finn e-book canonical document's extraction.

to apply standard image restructuring algorithms. Our approach is innovative because it proposes a complete physical restructuring process that is only based on the textual primitives contained in the electronic documents.

Actually, a unique physical segmentation of a document given a specific definition of region's homogeneity exists. However, in practice, one could get multiple interpretations due to a fuzzy or poor homogeneity criterion. For example, adjacent columns that are close to each other may be considered as a single block if the homogeneity measure used is not able to differentiate them, based on the spacing and alignment between the two blocks. The main difficulty of this chapter is precisely to find the right recipe leading to an accurate and homogeneous physical structure.

Concerning the physical restructuring algorithm itself, we used only a restricted set of textual primitive properties, i.e., unicode value, font size, character width, position, and font name. This choice has been made for two main reasons. Firstly, PDF does not ensure that white space, interline, and other typographical operators are correctly used, or even used. Secondly, our physical restructuring system is thus fully functional when applied on electronic document coming from scanned and OCR-ized hard copies. Indeed, such systems often provide only the basic and essential glyph properties. Consequently, *our system is designed to restructure electronic documents as well as OCR-ized paper documents by using only electronical textual primitives.*

Each restructuring step of our algorithm brings its own innovations. For instance, cleaning the raw PDF text primitives is a basic pre-processing step we introduced in order to

« Deluge » (détail). DAVID LACHAPELLE

LaChapelle décrypté

Odon Vallet, historien des religions, commente, dans une **vidéo**, *Deluge* et *Cathedral* du photographe américain David LaChapelle, dont une rétrospective a lieu à la Monnaie de Paris jusqu'au 31 mai : *« Cela peut évoquer des tableaux de Michel-Ange. On est devant une interprétation symbolique de la Bible. »*

Israël : revue de partis...

Les Israéliens ont voté, mardi 10 février, pour élire leurs députés. Pour tout savoir des forces politiques en présence, le **blog « Guerre ou paix »** passe les différents partis à la loupe.

... et portraits de « Refuzniks »

En Israël, on appelle « Refuzniks » ceux qui refusent de servir dans l'armée. Refus qui peut se traduire par de l'emprisonnement. Le photographe Martin Barzilai a rencontré certains de ces jeunes Israéliens réfractaires (pacifistes, activistes politiques, soldats marqués par la guerre). Un **visuel interactif** présente une galerie de portraits.

Films de guerre à Berlin

Envoyé spécial du *Monde* à la Berlinale, Jacques Mandelbaum analyse, à travers un **portfolio sonore,** quatre longs métrages qui sont à l'affiche au festival de Berlin (*Petits Soldats*, *Dans la brume électrique*, *Sturm* et *Le Messager*) et qui ont en commun d'évoquer les conflits actuels sans montrer les combats – des « films de l'arrière ».

Rendez-vous

Mercredi 11 février 2009, à 15 heures, **débat en direct** avec Eloi Laurent, économiste senior au département des études de l'OFCE (centre de recherche en économie de Sciences Po) : *« Crise globale : coordination internationale ou montée du protectionnisme ? »*

Nouvelle pauvreté et rupture du compromis social au Japon

Elle est pimpante, mais le message dont elle est porteuse est nostalgique : la Tokyo Tower, inspirée de la tour Eiffel, a été le symbole du redressement du Japon. Construite partiellement avec de l'acier provenant de tanks américains, légèrement plus haute que celle de Paris, elle fut achevée en 1958. Dominant une ville renaissant de ses cendres, elle était l'image de l'expansion et du retour de la fierté nationale.

Aujourd'hui, le ciel de la capitale est parsemé de gratte-ciel et la Tokyo Tower n'est plus porteuse de rêves. Dans les parcs alentour se nichent des sans-abri, avec leurs cartons et leurs hardes. Ils ne sont que la partie émergée de l'iceberg d'une nouvelle pauvreté.

Il est rare, dans la « société toboggan », que les perdants deviennent les gagnants. L'inverse est fréquent

La crise mondiale aggrave les inégalités existantes, qui se traduisent par une paupérisation, certes relative dans un pays riche, mais néanmoins alarmante. Entamé par les dix années de récession à la suite de l'éclatement de la « bulle spéculative » du début de la décennie 1990, le « compromis social » du Japon de l'expansion, fondé sur la recherche d'un équilibre entre compétition et solidarité, est désormais bel et bien rompu. Au-delà du brouhaha médiatique sur le Japon qui consomme et qui gâche réapparaît un Japon parcimonieux ; un Japon du vivre chichement des gagne-petit, surnageant au seuil de la détresse qui jette les plus démunis à la rue.

Les statistiques donnent le tableau de cette nouvelle pauvreté : 10 millions de salariés du secteur privé gagnent moins de 2 millions de yens (16 000 euros) par an. Et 20 millions – 35 % de la population active contre 20 % il y a quinze ans – occupent des

Analyse

Philippe Pons
Correspondant à Tokyo

emplois à contrat à durée déterminée (CDD), parmi lesquels les intérimaires sont près de 4 millions, soit trois fois plus qu'en 1997. En 2007, des raisons économiques ont été mises en avant pour expliquer le suicide de 7 300 personnes. Et, cette année, avant la fin mars, selon les chiffres du ministère de la santé et du travail, 124 800 travailleurs en CDD auront perdu leur emploi.

La société japonaise est désormais stratifiée en deux camps : les gagnants et les perdants. Passer du second au premier est rare. En revanche, l'inverse est fréquent : un phénomène que Makoto Yuasa, qui organise un mouvement de sans-abri, a baptisé la *« société toboggan »*. Au Japon, la formation s'acquiert surtout dans le cadre de l'entreprise. Et passer de petit boulot en petit boulot ne permet guère de se spécialiser : quand on est précaire à 20 ans, on a de fortes chances de le demeurer. La société nippone a toujours été compétitive, mais dans le passé elle était mobile. Ce n'est plus le cas.

La paupérisation est désormais un thème favori de la grande presse. Selon le quotidien *Asahi* (centre gauche), *« la déréglementation a été trop loin, et le recours à des travailleurs vulnérables détachés a touché un trop vaste secteur industriel »*. Au détriment de la sécurité de l'emploi – une des bases du consensus social des années 1960-1980 –, les entreprises ont joué la flexibilité en multipliant les CDD, qui sont les premières victimes des ajustements. Ceux qui restent travailleront davantage, avec, à la clé, du stress et du surmenage.

Les entreprises se sont en revanche constitué des réserves de main-d'œuvre : *« Quelle peut être la fierté de gestionnaires qui ont accumulé ces réserves et chassent des salariés gagnant 1 000 yens (9 euros) de l'heure ? »*, interrogeait récemment, au Parlement, Kaoru Yosano, le ministre des affaires économiques et financières.

Cette déréglementation, qui a creusé l'écart entre un secteur productif tourné vers l'exportation et un autre à la traîne, entre les villes qui concentrent la richesse et des campagnes qui se dépeuplent, n'a pas été compensée par un renforcement de la protection sociale. Seuls les emplois stables sont défendus par les syndicats, qui sont organisés, dans la plupart des cas, au niveau de chaque entreprise.

Le Parti conservateur, usé, tétanisé un temps par le néolibéralisme du premier ministre Junichiro Koizumi (2001-2006), qui « crucifiait » les « forces de résistance » critiquant le démantèlement du système de régulation sociale, a ignoré les faibles : les personnes âgées et les jeunes peu qualifiés, dont beaucoup de femmes, qui sont rejetés vers le travail intérimaire.

Au cours de la décennie « perdue » qui suivit l'éclatement de la bulle financière, le Japon a étalé dans le temps les réajustements, amortissant jusqu'à un certain point le coût social de la refonte de son appareil productif. Contrairement aux Etats-Unis, il n'a pas sacrifié le futur à la croissance : l'Etat a massivement emprunté auprès des Japonais, non de l'étranger. Mais la crise mondiale a remis au premier plan des problèmes qui étaient jusqu'alors négligés : la paupérisation, que les gouvernants pensaient résoudre *« naturellement »* par une croissance modérée, et la nouvelle façon d'aborder la question du travail.

Le « compromis social » nippon est rompu. Mais pas encore le lien social. L'Archipel conserve une stabilité enviable. Une foule de petites associations, des solidarités de voisinage ou de parenté et une endurance ancestrale, nourrie du sentiment que l'on ne peut compter que sur soi-même, pallient vaille que vaille la dureté du choc. Jusqu'à quand ? ■

Courriel : pons@lemonde.fr

Figure 5.23: The canonical document extraction performed on an extract of the french newspaper "Le Monde".

solve the poor PDF white space handling. The concept of α-layer introduced during the preprocessing phase is also an essential feature of our restructuring technique that allows the creation of simple merging and splitting algorithm able to handle any text orientation. Our dynamic thresholds generation, relative to text primitive font sizes, is also one of the key element explaining the success of our restructuring system. Our retroactive merging and post-linearization steps are completely original, they allow over-segmentation induced by text column justification to be recovered.

The top-down phase uses elaborated interline change detection and alignment change detection. These steps benefit from the generation of dynamic thresholds and some crucial syntactical properties in order to correctly detect paragraph's boundaries. Our phagocytosis step is also completely original, it recovers small text primitives that have not been merged with their parent text block because of their small font sizes.

Finally, the hybrid approach provides an efficient mean to recover the physical structure of any printed document, i.e., reconstruct its homogeneous blocks of text. The success of our methodology comes from the combination of our preprocessing, bottom-up, and top-down phases themselves divided into many critical steps.

Our canonical document extraction, and thus our physical structure analysis, would not be very useful without a concrete file format able to store accurately the extracted structures. Next chapter, i.e., Chapter 6, precisely exposes two XML-based canonical document formats: XCD and its optimized version OCD.

6

XCD & OCD, Two Canonical Document Formats

Contents

This chapter highlights two XML-based formats we developped in order to store canonical documents resulting from the physical restructuring of PDF documents in a human and machine readable format. Section 6.1 displays the motivations for the creation of "yet another file format". These motivations lead to the development of two XML-based canonical document formats: XCD and its optimized version OCD, presented in Section 6.2 and Section 6.3, respectively. Section 6.4 exposes the efficiency of the OCD file format in term of file size storage compared to PDF, XPS, and XCD. Finally, Section 6.5 concludes this chapter and presents potential future challenges.

6.1 Yet Another File Format

Over the years, various formats have been used to store document recognition results such as XDOC, DjVu, DAFS, METS/ALTO (see Subsection 3.6). Since our aim is to represent both the electronic primitives and their physical structures in a reliable way, none of these document formats were found to be suitable. Indeed, such formats are not able to represent vector graphics and font systems; they simply use text layers on top of raster images since they have been specifically developped to store scanned documents and their recognition results.

Existing electronic format such as ODA, RTF, SVG, or PDF have been studied in more details (see Subsection 3.6), since these file formats have the ability to reliably reproduce the original document layout. However, we are looking for a format that is fully compliant with our canonical document specification (see Chapter 5), and thus a canonical document format must fulfill the following set of criteria: universality, completeness, uniqueness, homogeneity, and simplicity.

ODA, RTF, PDF, and SVG suffer from irrevocable weaknesses since they do not fulfill the uniqueness, homogeneity, and simplicity criteria. Indeed, these file formats provide many different ways to represent similar graphical primitives which impede the uniqueness criterion to be fulfilled: Homogeneity is not respected either since textual elements are not intrinsically grouped into tokens, lines, and blocks. Concerning the simplicity criterion, it emphasizes the fact that a canonical document must be easily writable and readable, i.e., all information should be easily reachable and described thanks to precise, concise, and simple constructs. ODA, RTF, and PDF do not fulfill this criterion, because they all are complex and old document format represented with ASCII and binary streams. Despite its XML-based syntax, SVG is also an intricate document format containing an excessive number of XML constructs and, thus does not match the simplicity requirement either.

Many people use their own ad-hoc data formats with custom tools to visualize their recognition results. These file format tend to be very specific and unflexible. Consequently, none of the precited file formats was retained as potential candidate, because none of them match our well-defined requirements. Therefore we decided to design a new and complete electronic document format based on the XML standard.

In the paper entitled “Visualization of Document Recognition Results using XML Technology”, Hitz and Ingold [49] precisely emphasize the benefits of using XML-based format to store document recognition results. As well as the XML language, the physical structure of a document is a hierarchical construct which makes it an ideal format to store in a human-readable way the extracted canonical structure. Moreover, lots of writing, reading, and hanling tools dealing with XML descriptions are available. Transformation tool such as XSLT and XQuery [37] even allow source XML documents to be transformed into output XML documents. Let us note that XSLT can output any other kind of document format such as PDF and HTML, whereas XQuery is primarily used to query collections of XML documents (they can be accessed like databases). Consequently, and according to us, the conversion of document analysis results into a standalone XML format is definitely an ideal solution that allows further analysis to be performed and guarantees document’s content reuse.

6.2 XCD: an XML-based Canonical Document Format

XCD - XML Canonical Document - is an XML-based file format developed to reliably store the physical structures, content, and layout of restructured electronic documents. This section is divided into six subsections which precisely detail the internal representation of the XCD file format. Subsection 6.2.1 first presents the structure of the XCD file format. Then, Subsection 6.2.5, 6.2.2, 6.2.3, and 6.2.4 describe the representations of resources, text blocks, vector graphics, and raster images, respectively. Finally, Subsection 6.2.6, exposes the space coordinate system and device independence features of XCD.

6.2.1 XCD Structure

XML is a standard meta-language describing all kinds of structured or semi-structured data, such as documents. Creating a new XML file format is achieved by simply defining textual markup elements. Markup elements are embracing structures that can themselves be nested into other markup elements. Moreover, each markup element may contain textual data and an arbitrary number of attributes, i.e., key-value pairs. Markup elements follow precise lexical and syntactic rules specified by the XML language. An XML-based format is therefore intrinsically standardized, because it allows the XML data to be efficiently parsed and shared between various information systems.

Listing 6.1: A concise DTD defining the XCD file format (attributes are omitted).

```
<!ELEMENT  document      (page+)>
<!ELEMENT  page          (fonts, clips, images, graphics, texts)>
<!ELEMENT  fonts         (font*)>
<!ELEMENT  clips         (clip*)>
<!ELEMENT  images        (imageblock*)>
<!ELEMENT  imageblock    (image+)>
<!ELEMENT  graphics      (graphicblock*)>
<!ELEMENT  graphicblock  (graphic+)>
<!ELEMENT  graphic       (line | cubic | quadratic)
<!ELEMENT  texts         (textblock*)>
<!ELEMENT  textblock     (textline+)>
<!ELEMENT  textline      (token+)>
```

Listing 6.1 presents a concise specification of XCD, in the form of a document type definition. Basically, an XCD file is an XML formalism storing a canonical document. Thus, an XCD file is composed of a list of pages, each containing its font descriptions, clipping paths, and graphical primitives, i.e., images, graphics, and text blocks. Moreover, each graphical primitive is characterized by a bounding box and has a reference to a clipping path. Additionally, each text primitive references its corresponding font.

6.2.2 Representation of Text Blocks

The XCD file format respects carefully the five canonical document rules; thus a special care is dedicated to textual content. Listing 6.2 presents an extract of the canonical file

corresponding to the "tiny PDF drawing" example (see Figure 4.3), in order to demonstrate how XCD represents text blocks, lines, and tokens.

Listing 6.2: The XCD text block corresponding to the "tiny PDF drawing" example.

```
<textblock bounds="83.3 78.8 46.3 25.3">
  <textline>
    <token clip-id="1" z-order="1" font-size="12.0" font-name="Helvetica"
        stroke="0 0 0 0" fill="0 0 0 255" type="text" coords="83.3 87.4 86.3
        87.4 89.3 87.4 96.1 87.4 102.1 87.4" content="116 105 110 121"/>
    <token clip-id="1" z-order="1" font-size="12.0" font-name="Helvetica"
        stroke="0 0 0 0" fill="0 0 0 255" type="space" coords="102.1 87.4
        105.1 87.4" content="32"/>
    <token clip-id="1" z-order="5" font-size="12.0" font-name="Helvetica"
        stroke="0 0 0 0" fill="0 0 0 255" type="text" coords="105.1 87.4 113.3
         87.4 122.3 87.4 129.6 87.4" content="80 68 70"/>
  </textline>
  <textline>
    <token clip-id="1" z-order="9" font-size="12.0" font-name="Helvetica"
        stroke="0 0 0 0" fill="0 0 0 255" type="text" coords="86.7 101.6 93.5
        101.6 97.2 101.6 104.0 101.6 113.0 101.6 116.0 101.6 122.7 101.6 129.4
         101.6" content="100 114 97 119 105 110 103"/>
  </textline>
</textblock>
```

We observe that the text content is organized in text blocks containing text lines themselves composed of tokens respecting the reading order. Each text block defines a bounding box thanks to the *bounds* attribute. Additionally, each token references a clipping path *clip-id* and a font *font-name*. Font sizes, fill and stroke colors, text types, z-orderings, and character coordinates are expressed thanks to *font-size*, *stroke*, *fill*, *type*, *z-order*, and *coords*, respectively. The coordinates are expressed as a sequence of absolute x- and y-coordinates, beginning at the first character and ending after the last character. Finally, the text content is described in the *content* attribute, using unicode values expressed with integer numbers.

6.2.3 Representation of Vector Graphics

The XCD format supports vector graphics, they are described using only line segments, cubic and quadratic Bézier curves. Indeed, thanks to these three path primitives, it is possible to generate any kind of vector graphics accurately. Path primitives are then grouped together in order to form graphic objects, themselves characterized by various attributes such as *stroke* and *fill* colors, *stroke-width*, *fill-rule*, *transform* matrices, etc.

Listing 6.3: The "tiny PDF drawing" blue ellipse represented in XCD. The ellipse is composed of four cubic Bézier curves.

```
<graphicblock bounds="72.0 733.4 72.0 36.6">
  <graphic clip-id="1" z-order="0" stroke="0 0 0 0" fill="204 204 255 255"
      stroke-width="0.1" cap-style="butt" join-style="round" dash-array="0.0"
      dash-phase="0.0" fill-rule="evenodd" transform="1.0 0.0 0.0 -1.0 0.0
      842.0">
```

```
    <cubic x0="108.0" y0="733.4" x1="88.1" y1="733.4" x2="72.0" y2="741.6" x3=
        "72.0" y3="751.7"/>
    <cubic x0="72.0" y0="751.7" x1="72.0" y1="761.8" x2="88.1" y2="770.0" x3="
        108.0" y3="770.0"/>
    <cubic x0="108.0" y0="770.0" x1=127.9" y1="770.0" x2="144.0" y2="761.8" x3
        ="144.0" y3="751.7"/>
    <cubic x0="144.0" y0="751.7" x1="144.0" y1="741.6" x2="127.9" y2="733.4"
        x3="108.0" y3="733.4"/>
  <graphic>
</graphicblock>
```

Graphic objects are often used for embellishing documents' background, so they may sometimes be overlayed. Although XCD does not include the concept of layers, the rendering order of fore and background objects is revealed by a *z-order* parameter. Listing 6.3 presents an XCD extract corresponding to the blue ellipse shown in the "tiny PDF drawing" example.

6.2.4 Representation of Images

Raster graphics support is also provided by the XCD format. Digital images are simply stored in external files (*file* attribute) using the lossless PNG image file format as shown in Listing 6.4. The image position and size are specified by a simple affine transform matrix.

For instance, in Listing 6.4, the *transform* matrix contains the following values "0.6 0.0 0.0 0.6 410.8 212.7", meaning that the image has a uniform scale of 0.6 (in x- and y-directions) and a position (or translation) of x- and y-coordinates equal to (410.8, 212.7).

Listing 6.4: In XCD, an image is represented with absolute x- and y-coordinates and a relative path pointing to a PNG file.

```
<imageblock id="0" bounds="411.7 213.3 300.9 318.4">
  <image clip-id="3" z-order="1" transform="0.6 0.0 0.0 0.6 410.8 212.7" file=
      "./image.png"/>
</imageblock>
```

6.2.5 Representation of Clipping Paths and Fonts

The fonts and clipping paths contained in a canonical document use the same XML formalism as do vector graphics. Thus, a clipping path is simply described by a closed path and referenced by means of a *clip-id*, i.e., a clip identification number.

A font has a more elaborate structure: it contains a set of attributes such as *font-name*, *font-id*, *ascent*, and *descent*. Moreover, a font embeds a list of glyphs, each of them described by a *path*, a *unicode* value, and a character *width* (in a unitary point-size).

6.2.6 Space Coordinates and Device Independence

XCD represents graphics coordinates in a two-dimensional cartesian coordinate system. Whereas the PDF file format defines a coordinate system whose origin is in the bottom left corner of a

page, the origin of the CD (and thus XCD) coordinate system is defined by the top left corner of a page, as do most of the standard document representation systems. The x-coordinate increases from left to right, while the y-coordinate increases from top to bottom.

Coordinates of text, graphics, images, and clipping paths are all expressed as absolute positions relative to the origin of the page's coordinate system. Device independence is then ensured by using vector graphics and high resolution images given a defined resolution in DPI - dots per inch -. Each coordinate is then expressed as a real number relative to this resolution: images, graphics, text positions and sizes. Colors are also independant of any device; they are expressed with integer values using the scRGB [116] color space (wide color gamut RGB color space), together with an integer alpha value ranging from 0 to 255 (0 being transparent and 255 opaque).

6.3 OCD: an Optimized Canonical Document Format

OCD - Optimized Canonical Document - is an evolution of XCD. A very unpleasant property of the XCD file format, that is in fact intrinsic to the XML format, is that it is quite verbose, leading therefore to excessive file sizes. Most of the time, this drawback of XCD makes no difference, but when original documents are large, XCD files become excessively huge. This drawback can lead to serious troubles, such as out of memory errors or worthless processing time. We effectively experienced these problems when working with big newspapers and e-books (see Tables 6.1, 6.2, and 6.3 in Section 6.4).

Thus, the OCD file format has precisely been developped in order to resolve the XCD space storage wasting. OCD still keeps the fundamental objectives of XCD: preserving the visual rendering of static documents while keeping a clean, canonical, and structured internal representation. But OCD solves the XCD over-sized file problem by removing information redundancy and optimizing the internal representation. Further, as described in the next sub-sections, OCD is a good trade-off between structuring, readability, and compactness.

This section presents the main lines about the OCD file format; a detailed specification is available in Appendix A. OCD is now the canonical document format we use, as a consequence XCD has become obsolete (reason why we do not detail its specification in a dedicated Appendix). This section is divided in five subsections; Subsection 6.3.1 first presents the structure of the OCD file format. Then, Subsection 6.3.2, 6.3.3, 6.3.4, and 6.3.5 describe the XML representation of the OCD resources, text, graphics, and images, respectively.

6.3.1 OCD Structure

The aim of OCD is to have a canonical, structured, compact, and easy to handle file format that preserves exactly the original document appearance. Similarly to XCD, three types of graphical objects exist in the OCD format: text, vector graphics, and raster images. Thus, the XML structure of OCD is quite similar to the XCD one. OCD uses the same coordi-

nate system and device independence mechanisms as XCD. However, while XCD expresses real values with floating point numbers using a 64 bit precision, OCD truncates them to a meaningful number of decimals (similarly to PDF), avoiding like this unnecessary huge floating-point number representations.

Listing 6.5: A concise DTD defining the OCD file format (attributes are omitted).

```
<!ELEMENT  ocd        (resources, pages)>
<!ELEMENT  resources  (fonts, clips, pool?)>
<!ELEMENT  fonts      (font*)>
<!ELEMENT  font       (glyph+)>
<!ELEMENT  clips      (clip*)>
<!ELEMENT  pool       (g+)>
<!ELEMENT  g          (image | path | text | g)+>
<!ELEMENT  pages      (page+)>
<!ELEMENT  page       (image | path | text | g)*>
```

Listing 6.5 shows a concise specification of the OCD format in the form of a DTD. Contrary to XCD, font definitions as well as clipping path descriptions are handled as document resources. Similarly, redundant text, images, and graphics can also be embedded as resources in order to avoid multiple descriptions of the same graphical primitive. Note that OCD uses far more compact data descriptions than XCD. Furthermore, in opposition to XCD, OCD is a complete and standalone file format since all data are embedded in a single file, i.e., external resources are strictly forbidden.

The compactness of an OCD file is mainly achieved through information redundancy removal and XML character stream compression, thanks to the GZIP deflate standard, i.e., a combination of the LZ77 algorithm and Huffman coding. Note that, specific XML data compression algorithms exist, such as XMLPPM, DTDPPM, XMLZIP, XMill, or COMPREX [76, 104]. However, since none of these algorithms has emerged as an XML compression standard, we have chosen to use the GZIP compression because it is widely used and supported. Moreover, comparative tests tend to highlight the fact that the GZIP compression algorithm performs well even against specific XML compressors.

6.3.2 OCD Resources and Pool of Objects

The OCD file format begins with a description of the document resources, i.e., fonts, clipping paths, and graphics objects. The advantage of defining a set of resources is that all the pages of a document share them, thus avoiding the unnecessary duplication of information. For instance a font used in several pages of a document is defined only once; the same procedure works for clipping paths and can be applied to text, graphics paths, and raster images (e.g., recurrent text, background images, logo graphics, etc.). Note that font glyphs, clipping paths, and graphics paths use a common vector graphics formalism (see Subsection 6.3.4).

Concerning graphics objects, they are put together in a pool of objects and refererenced by means of *g* IDs ("g" stands for "group", meaning that one or several graphics objets are

grouped together). Fonts and clips are also referenced by means of IDs: font IDs and clip IDs, respectively.

6.3.3 Text Representation

Text representation is the most critical part of our OCD file format; it is also the most elaborate one. In OCD, text representation greatly benefits from our canonical text blocks generation. Indeed, since canonical text blocks are homogeneous blocks of text, their description needs very little information.

In XCD, text blocks do not take advantage of information redundancy. Each text attribute is written in the output file, whereas positions are expressed thanks to absolute coordinates. On the opposite, OCD is far more space saving, e.g., it uses relative positioning and does no more store redundant data.

In OCD, each single page contains a unique internal graphics state that is used for texts, images, and graphics. This technique avoids a lot of information redundancy since a graphical property is explicitly written only if its value has changed. In OCD each token is represented as a sequence of unicode characters thanks to hexadecimal values. Positions are computed for each character glyph thanks to the current graphics state: character space (*cs*) and white space (*ws*) are combined with the character width (available in the font resources) and the current transform matrix. Token and line x- an y-offsets are expressed thanks to dedicated attributes: *tx*, *ty*, *lx*, *ly*, respectively.

Listing 6.6: An OCD text block corresponding to the "tiny PDF drawing" example. The text content is organized in text blocks containing text lines themselves composed of text tokens respecting the reading order.

```
<g type="block">
  <g type="line">
    <text x="33.6" y="39" scale="12" font="Helvetica" fill="0" cs="4 -5 1 0">
        74 69 6e 79</text>
    <text ws="282"/>
    <text cs="0 -5 0">50 44 46</text>
  </g>
  <g type="line">
    <text lx="3.6" ly="14" cs="10 -8 10 -5 -5 10 0">64 72 61 77 69 6e 67</text
        >
  </g>
</g>
```

Listing 6.6 shows the OCD text block corresponding to the "tiny PDF drawing" example. In this listing, we observe that the OCD file format represents text blocks with nested <*g*> markup elements which group together text content into lines and blocks. That is, the outer <*g*> markup element groups lines together and declares an attribute *type* equal to "block". The second and nested <*g*> markup element groups text tokens together into lines and declares an attribute *type* equal to "line".

The actual content of a text token is handled by the <*text*> markup element. Its CDATA contains hexadecimal values representing character unicodes whereas its attributes describe the text properties. For instance, clipping path ID, font ID, fill and stroke colors are represented with the following attributes: *clip*, *font*, *fill*, and *stroke*, respectively. Special attributes called *scale* and *shear* are used to express the transform matrix. x and y are used as absolute x- and y-coordinates whereas *lx*, *ly*, *tx*, *ty*, *cs*, and *ws* are used to update line offset, line spacing, text offset, text rise, char spacing, and white spacing, respectively. Concerning the *cs* (char space) attribute, its value is a sequence of real numbers which is trimmed in case of sequential recurrent data. Again, note that OCD attributes are used in conjunction with the page graphics state and, as such, they are explicitly written only when their values are modified.

6.3.4 Graphics Representation

In XCD each path element is represented with a <*path*> markup element, itself containing sub-path markup elements such as <*line*>, <*cubic*>, and <*quadratic*>. These XCD elements are described thanks to absolute coordinates. On the opposite, in OCD, a path is represented in a much more compact way, quite similar to an SVG path.

Indeed, in OCD, a unique <*path*> element is used, sequential sub-paths are then described in the markup element character data by using predefined operators and operands expressed thanks to relative coordinates. Existing sub-path operators are *m*, *l*, *c*, *q*, and *z* standing for *move*, *line*, *cubic*, *quadratic*, and *close*, respectively. The *m* operator moves to the beginning of a path (or sub-path) at some coordinates without drawing, whereas the *z* operator closes a path by drawing a straight line back to the last *m* operator. Then, *l*, *c*, and *q* are used to draw a line, a cubic Bézier curve, and a quadratic curve, respectively. Cubic and quadratic curves allow complex and non-linear paths to be represented accurately.

Listing 6.7: The "tiny PDF drawing" blue ellipsis represented in OCD. The ellipsis is composed of four cubic Bézier curves.

```
<path x="0" y="85" fill=".8 .8 1">m 58.3 -24.9 c -19.9 0 -36 -8.2 -36 -18.3 c
    0 -10.1 16.1 -18.3 36 -18.3 c 19.9 0 36 8.2 36 18.3 c 0 10.1 -16.1 18.3
    -36 18.3 z</path>
```

Each operand x- and y-coordinates are relative to the previous coordinates (i.e., they are delta values), except for the initial *move* operator of a path. Thus, every OCD path begins with an absolute *move* operator. Listing 6.7 shows the OCD representation of the "tiny PDF drawing" blue ellipsis.

6.3.5 Images Representation

In XCD, raster images are represented thanks to PNG files referenced by relative file paths. However, the OCD file format does no more allow external resources such as image files.

Fortunately, since an OCD file is a compressed ZIP archive, images are naturally embedded in OCD as zipped image files. Lossless image compression is handled with the PNG file format, whereas lossy image compression with JPEG files. Listing 6.8 shows the representation of an image object referencing a PNG file embedded in an OCD file.

Listing 6.8: Images are directly embedded in the OCD ZIP archive as PNG or JPEG files and referenced by means of IDs.

```
<image id="image1.png" width="473" height="344" x="120" y="48" scale="0.8"/>
```

6.4 OCD Performances

The aim of OCD is to have a canonical, structured, compact, and easy to handle file format that preserves exactly the original document appearance. Since the canonical and structured properties are ensured by the canonical document itself, our evaluation has focused on the compactness of OCD.

OCD has been compared to two electronic document standards, i.e., PDF from Adobe and XPS from Microsoft. XCD file sizes have also been reported in our evaluation in order to grasp the storage improvements of the OCD file format. Our tests have been made on three different classes of documents, i.e., e-books, graphical documents, and newspapers.

Several non-tagged PDF e-books have been downloaded from PlanetPDF [85] and exported to XPS, XCD, and OCD. These e-books contain nearly only textual primitives, thus, it is possible to grasp the compactness of the OCD textual encoding scheme (see Table 6.1).

Document title	PDF	XPS	XCD	OCD
Aesop's Fables - 93 pages	243 KB	441 KB	7'014 KB	91.9 KB
Around the World in 80 Days - 339 pages	766 KB	1'912 KB	36'045 KB	422 KB
The Odyssey - 550 pages	1'280 KB	3'082 KB	63'693 KB	850 KB
The Last of the Mohicans - 698 pages	1'605 KB	3'944 KB	80'896 KB	924 KB
Ulysses - 1305 pages	2'953 KB	7'334 KB	-	1'743 KB
Mean PDF file size ratio	100%	233%	4402%	55%

Table 6.1: A comparison of e-books file sizes. In average, the OCD size is 55% of the PDF size. Concerning the last XCD file, it caused an out of memory error.

Afterwards, we looked at graphical documents containing a lot of vector graphics since they are a good testbed to determine the graphical encoding performances of our OCD file format (see Table 6.2). City maps and cadastre have been acquired directly from official Internet city web sites. The testbed also contains a music sheet and a tiger drawing (see Figure 6.1) known as the "SVG Tiger".

Document title	PDF	XPS	XCD	OCD
The SVG Tiger - 305 paths	32.2 KB	57.3 KB	280 KB	27.1 KB
Mario Bros Music Sheet - 812 paths	130 KB	278 KB	2'888 KB	64.8 KB
Ascona City Map - 3981 paths	822 KB	606 KB	4'812 KB	368 KB
Manhattan Bus Map - 4571 paths	265 KB	-	5'264 KB	252 KB
Estavayer Cadastre - 107799 paths	1'544 KB	1'818 KB	21'924 KB	1'306 KB
Mean PDF file size ratio	100%	146%	1417%	72%

Table 6.2: A comparison of vector graphics file sizes. In average, the OCD size is 72% of the PDF size. Note that the original SVG Tiger file has a size of 88.1 KB. Concerning the fourth XPS file, the Microsoft XPS printer was not able to render it correctly.

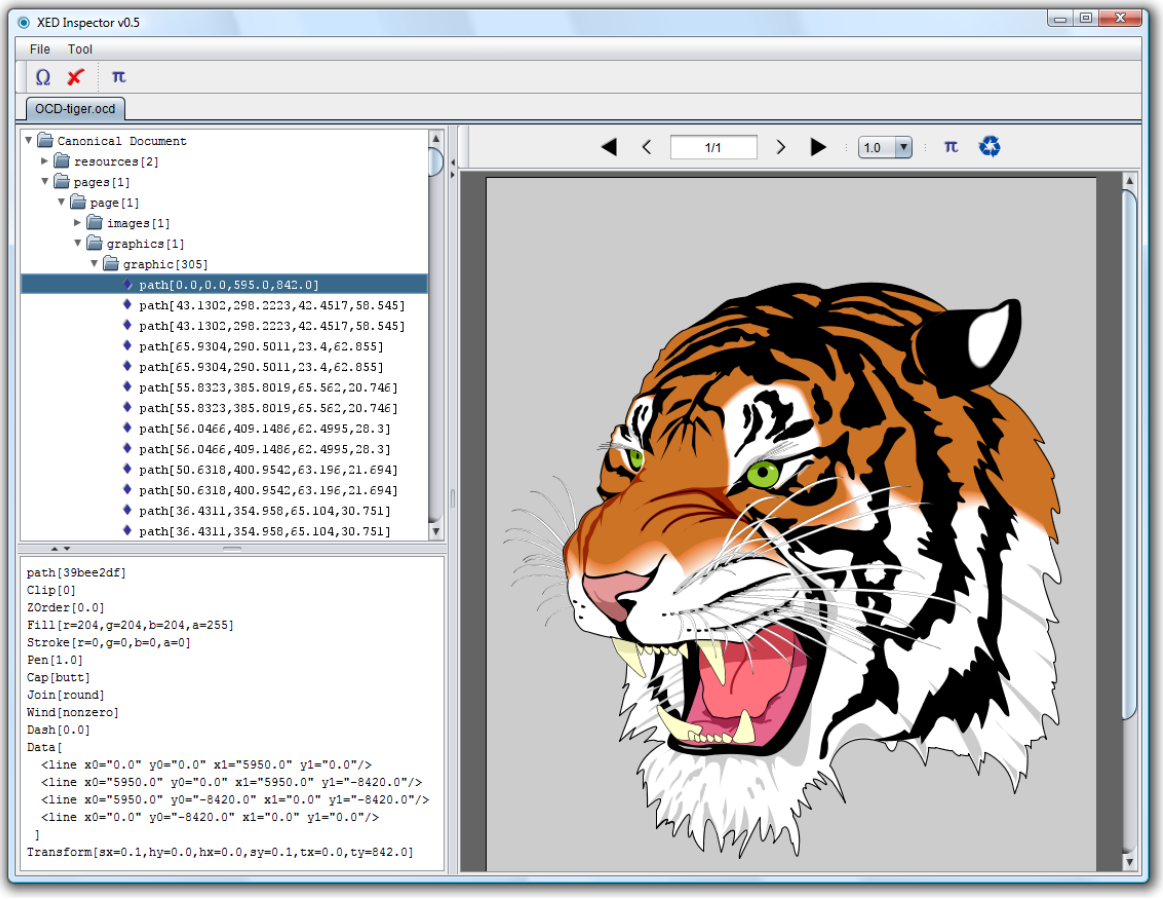


Figure 6.1: The SVG Tiger is entirely composed of vector graphics, the original SVG file has first been exported to PDF, then stored in OCD, and finally openened in XEDInspector.

Finally, we evaluated the OCD file size of newspaper documents, since they have complex document structures containing lots of text, graphics, and images. Images in OCD are encoded using PNG and low-compressed JPEG. The compactness of newspaper files is less relevant than the two previous evaluations. Actually, PDF uses several lossy and lossless compression schemes; therefore, the original PDF images have to be processed and normalized into standard scRGB JPEG and PNG images. Nevertheless, the newspaper class of documents reflects the general encoding performances of the OCD file format (see Table 6.3).

Document title	PDF	XPS	XCD	OCD
The Boston Globe Frontpage - 2004/08/24	391 KB	190 KB	2'990 KB	175 KB
The IHT Frontpage - 2005/04/11	299 KB	1577 KB	3'697 KB	253 KB
La Liberté - 2009/01/16	5'422 KB	8'899 KB	77'619 KB	4'843 KB
Le Monde - 2009/01/16	9'093 KB	12'379 KB	106'496 KB	10'664 KB
Metro Santa Cruz Weekly - 2009/04/15	38'272 KB	38'912 KB	-	30'310 KB
Mean PDF file size ratio	100%	196%	1151%	83%

Table 6.3: A comparison of newspapers file sizes. In average, the OCD size is 83% of the PDF size. Concerning the last XCD file, it caused an out of memory error.

Tables 6.1, 6.2, and 6.3 show that even though OCD is a strongly structured and XML-based format, its compactness is very efficient compared to PDF and XPS. OCD improves PDF compactness when dealing with textual or/and graphical documents while still structuring their content. Indeed, a canonical document ensures that textual primitives are organized in homogeneous blocks of text, thus allowing information redundancy to be greatly reduced, leading to a better compression. Moreover, redundant graphical primitives are described only once in the document resources. Table 6.1 shows that the evaluated OCD textual documents are almost two times smaller than the PDF ones, while Table 6.2 shows that the evaluated OCD graphical documents are less than three quarter the size of the PDF ones.

The Microsoft XPS file format seems to give acceptable results. Actually, XPS does not ensure the preservation of the original PDF visual information: raster images are regularly down-sampled when printed in XPS. Worst, as shown in Figure 6.2, vectorial graphics are sometimes just replaced with low-resolution raster images by the XPS writer, as it is the case for the Manhattan Bus Map (see Table 6.2).

It is important to notice that the PDF file format may embed non visual information such as annotations. PDF may also be incrementally saved by adding new information at the end of the file while keeping old and worthless data. PDF streams and therefore content streams may even be left uncompressed. In our case, all the evaluated PDF files always used stream compression and none of them used incremental updates facilities. Nevertheless, few PDF annotations still remain; thus the file size comparison between PDF, XPS, XCD, and OCD must be appreciated with care.

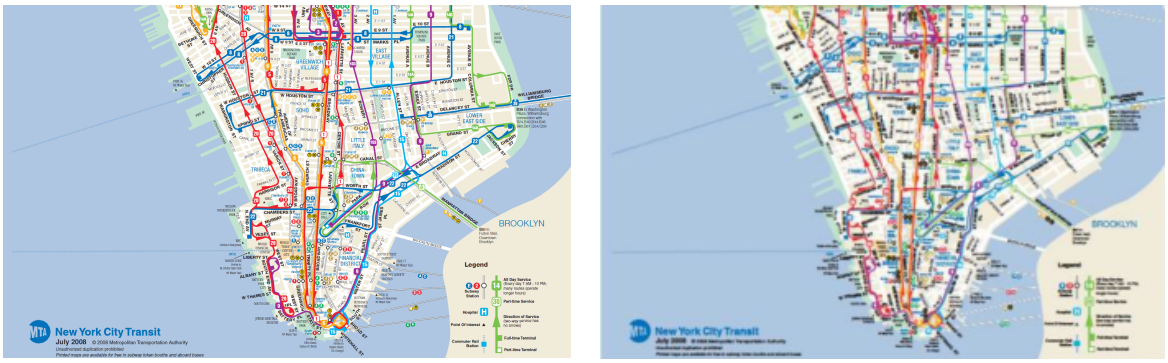


Figure 6.2: On the left, a piece of the original vectorial PDF Manhattan bus map, on the right, the same piece of map rasterized by the Microsoft XPS writer.

6.5 Conclusion

This chapter has presented XCD and OCD, two XML-based file formats developped to store the content, layout, and structure of electronic documents. Our evaluation showed that OCD greatly improves the performance, in term of storage size, of XCD, our first canonical document format. The file size reduction from XCD to OCD is very impressive, about a factor of 20. Indeed, XCD files are not optimized and do not use any compression algorithm. On the opposite, OCD optimizes its storage size thanks to the following features:

- In OCD, text and graphics are represented with concise descriptions that avoid redundant information;
- Real values are truncated to a meaningful number of decimals, thus avoiding unnecessary huge floating-point number representations;
- At a higher level, OCD provides the ability to store identical graphical primitives only once as resources of a document;
- The OCD file size is also greatly reduced thanks to the compression of the XML data stream by using the GZIP standard.

According to our evaluation, the average OCD file size is even better than the average PDF file size. Three reasons may explain such performances. First, PDF files can't be entirely compressed, since only PDF content streams support data compression. Then, as exposed in the previous section, PDF may embed invisible metadata such as annotations. Nevertheless, annotations alone can't explain the significant file size reduction of OCD compared to PDF. The final reason, and certainly the most important one, is that a) OCD uses a mechanism of resources and pool of objects that avoids multiple descriptions of recurring data, and b) OCD

uses compact and non redundant descriptions, benefitting from the homogeneous canonical text blocks extracted during the physical restructuring phase (see Chapter 5).

OCD provides electronic document facilities but does not address results coming from OCR recognition system. Indeed, a complete document analysis format should provide low-level features such as OCR recognition facilities. OCR results may provide fuzzy solutions expressed with character confidence and alternate possibilities.

However, OCD is a simple XML-based format; it is therefore easy to add new functionalities. For instance, OCR results could be expressed thanks to two new attributes: a first one called *confidence*, used to express the recognition probability of a given character, and a second one called *alternate*, used to express potential alternate possibilites.

Note that OCD does not rely on any PDF internal structures or representation; in this manner an OCD file can easily be generated and used by any document processing software. OCD stores the content, layout, and physical structure of electronic documents, however, high-level features such as the logical structure should also be taken into account. This issue is precisely addressed in the next chapter, i.e., Chapter 7, which focuses on the logical restructuring of OCD documents and their storage in OCD+, a logically restructured version of OCD.

7

Dolores, an Interactive Tool for the Logical Restructuring of PDFs

Contents

This chapter presents Dolores, a windowed environment that aims at recovering the logical structures from canonical documents by interactively inferring their document models. As shown in Chapter 5, a key aspect of our canonical document formalism is precisely to provide a pivot format facilitating high-level processing such as the recovering of the logical structure from electronic documents.

The outline of this chapter is described below. First, in Section 7.1, the reasons leading to the creation of Dolores are explained: we highlight the needs for new and flexible logical restructuring systems. Then, Section 7.2 emphasizes the philosophy that guided us during all the development of Dolores. Section 7.3 presents the interface of Dolores, its dynamic and interactive labeling environment. Section 7.4 dissects the learning system integrated in Dolores which inherits from the standard multilayer perceptron (i.e., a feedforward artificial neural network). Finally, Section 7.5 underlines the achievements of Dolores and some ideas for potential improvements and extensions.

7.1 Needs for Flexible Systems

Nowadays, few document logical restructuring environments have come out. The recognition of the logical structure is a complex task that has to be adapted to specific classes of doc-

uments and applications. Thus, it is not possible to define a universal logical restructuring process, unlike physical structure recognition. Therefore, an analysis system needs some contextual information about its targeted set of documents in order to modelize the relations between their physical and logical structures, i.e., the document model.

Hence, analysis systems that aim at recovering some logical structure from electronic documents (PDFs in particular) all tackle the same inevitable issue: the creation of document models. Unfortunately, most of the existing systems use document images as input and so loose the accessible content information of electronic documents. Still, some systems taking advantage of PDF primitives exist, but none of them does provide a generic solution, since they all target at customized restructuring tasks for very specific sets of documents [69, 6, 94, 28, 94], e.g., recognizing the table of contents of the "Times" magazine.

Worse, except for some previous experiments done in our research group by Lyse Robadey [97], none of the existing systems provides functionalities such as interactive learning environments and assisted acquisition of document models. Most of the existing approaches generate their document models either manually or automatically (without human interaction). As one can suppose, manually editing a document model is a laborious task that has to be done by an expert user. On the opposite, automatic systems infer their document models directly from large sets of ground-truthed data; nevertheless, such data has to be manually acquired during a preliminary labeling phase, which is again a demanding task. Furthermore, automatically inferred models tend to be static since they are tightly coupled to their ground-truthed data, i.e., they are unable to cope with unknown situations since it is impossible to modify them (unless the creation of a new set of ground-truthed data).

To put it in a nutshell, the creation of document models is one of the most time consuming tasks when researchers and end-users want to recover structures from documents: either the document model itself or the ground-truthed data has to be manually edited. In Chapter 3, "State of the Art", we precisely highlighted the fact that there is a real lack of flexible tools like assisted environments with user-friendly interfaces, visual feedback of users' actions, recognition improving with iterative use or models being refined incrementally. As a consequence, our researches led to the development of a complete and innovative document logical restructuring system called Dolores, which is able to produce document models thanks to a dynamic user interface providing interactive and incremental learning functionalities.

7.2 Dolores' Philosophy

Human beings are able to cope with new situations by using their knowledge; they learn through experiences and improve their own knowledge incrementally. In document analysis, new situations are also inevitable because there is an infinite variety of documents, and therefore, it is unconceivable to develop a universal system able to recognize any document structure a priori. Based on this assumption, a strategy similar to the "human way of

learning" should be applied to the recognition of document structures. Accordingly, an incremental learning mechanism based on an interactive system (used to gain experience) seems the right path to follow.

In this sense, Dolores subscribes to the CIDRE - Cooperative and Interactive Document Reverse Engineering - [11] philosophy which precisely advocates the idea that an analysis system does not work in a fully automatic way, but cooperates with the user. Thus, Dolores provides a quite simple human-computer interaction: the learning system first proposes a solution, second, the human performs some correction, and third, the system analyses the correction and adapts its model consequently, in order to propose a better solution (the learning process is therefore iterative).

This human-computer collaboration is achieved through interactive tools and advanced visualizations, it has various advantages for both users and systems. On the one hand, interactions increase the efficiency of the system incrementally by learning from users' feedbacks, and dynamically by adapting itself to the document characteristics. On the other hand, users take full benefit from the visualization, by rapidly retrieving and correcting analysis errors. Furthermore, interacting with the graphical representation of analysis results allows the system to be dynamically tuned at a high-level, without manually editing intricate configuration files or parameters.

The CIDRE philosophy has been successfully applied in 2(CREM) [97], where the users can supervise the analysis of logical structures using the XMIllum graphical user interface [48]. Dolores is inspired by 2(CREM) and thus connects together an incremental learning system and an interactive tool. In practical terms, the idea is to provide an intuitive and interactive interface able to dynamically assign logical labels to the canonical document's text blocks. Thus, the user actively interacts with the interface of Dolores by labeling a canonical document (extracted from a PDF with XED) while the underlying learning system infers a model itself used to speed-up and simplify the following actions of the user. Then, once a stable model has been defined, the user can interactively apply it to other canonical documents belonging to the same class of documents. Further, the user can decide at any moment to improve or tune the document model by interacting again with the learning facilities of Dolores. Last, note that the interface of Dolores has to be intuitive and friendly; therefore, all the learning system complexity and processing has to be "left behind the curtain".

7.3 Dolores, an Interactive Learning Environment

Dolores (for Document Logical Restructuring) is an interactive analysis system with learning capabilities designed to recover the logical structures from textual documents such as newspapers, scientific papers, journals, magazines, e-books, etc. Dolores is entirely based upon the canonical document format, used to store the content and layout of static documents, which has precisely been elaborated for further high level processing, i.e., logical structure

reconstruction and document understanding.

Dolores integrates a complete and innovative learning system, supported by a graphical user interface, allowing document models to be inferred from interaction. As shown in Figure 7.1, the interface of Dolores is composed of two complementary windows:

- The *labeling window* is the core of the interface: it provides interactive labeling facilities and assists the user thanks to visual aids; i.e., the user interactively improves the document model first by verifying the color label assigned to each text block and then by correcting the potential recognition mistakes.

- The *document model window* displays a visual representation of the document model which is also the learning system integrated in Dolores, i.e., an innovative neural network based on a multilayer perceptron (MLP) which reveals its internal knowledge using visual properties relative to a "relevance criterion".

Both the internal working of Dolores and its interface are detailed in the rest of this chapter. The following subsections describe three key aspects of Dolores: a) its labeling interface (Subsection 7.3.1), b) its document model (Subsection 7.3.2), and c) the set of extracted features used to feed its learning system, i.e., infer its document model (Subsection 7.3.3). A special attention is dedicated to the elaborate learning system of Dolores. Therefore, it is exposed in a standalone section. Section 7.4 presents a custom-made neural network which aim is twofold: first, facilitating its use by removing any manual settings and, second unveiling the black box of standard neural networks by providing visual feedbacks about the knowledge stored in the network links.

7.3.1 Labeling Interface

As stated earlier, the aim of Dolores is to infer document models thanks to an assisted environment allowing canonical text blocks to be interactively and dynamically labeled. The interface of Dolores displays canonical documents that have been extracted with XED during a previous physical structure reconstruction phase. Figure 7.2 precisely shows the labeling interface in Dolores which is composed of:

- an upper toolbar providing buttons to navigate through a canonical document;

- a bottom toolbar allowing OCD and XCD files to be opened as well as saved;

- a central page panel displaying the current canonical page;

- a popup contextual menu providing labeling functionalities.

Additionally, as it displays the page of a canonical document, the page panel also displays some meta information about it. In Figure 7.2, we observe that each text block is highlighted by its bounding rectangle. The color shade of each bounding rectangle corresponds to the

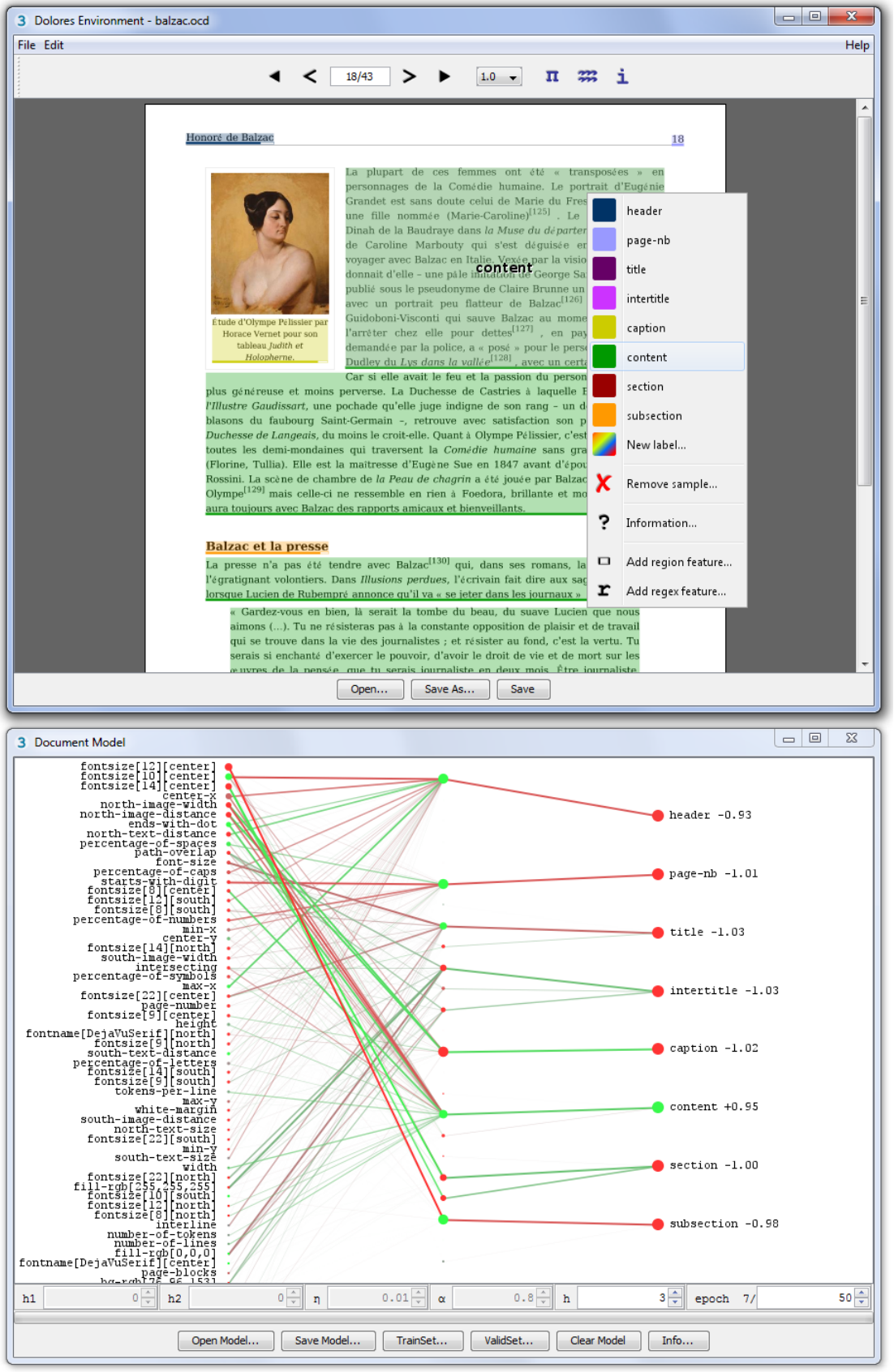


Figure 7.1: An OCD file opened in Dolores and logically labeled thanks to the interactive learning system. The interface is composed of two windows: a labeling window (on the left) and a document model window (on the right).

label assigned by the recognition system to the text block (labels and colors are defined by the user). If no label has yet been defined, i.e., the learning system has just been initialized, a default gray color is used. Concerning the stroke color (the surrounding rectangle), it is drawn only if a text block has been integrated in the training set. Therefore, by looking at the surrounding stroked rectangle, the user can instantly know whether a text block is part of the training set of the current document model or not. Let us note that the color shade and stroke of a text block may be different, meaning that the classification done by the system does not match the ground-truthed label, i.e., the recognition of the text block failed.

Figure 7.2: Interactive logical labeling of a canonical document. Surrounded text blocks mean that they have been integrated in the training set of the document model, whereas their color shades correspond to the labels assigned by the recognition system.

During the elaboration of a document model, logical labels may be created at any time thanks to the contextual popup menu of the interface. Indeed, at the initialization of a document model, no predefined labels are set; actually, the user creates its own label names as he wants and when he wants. A right click of the mouse inside a bounding box triggers a pop up window in which the user can either choose an existing label or define a new label to assign to the text block (see Figure 7.2). Then, once a label has been assigned, the document

model is updated in consequence: the labeled text block is first integrated in the training set, a training phase is then performed "on the fly", directly followed by a re-classification (i.e., labelization) of all the text blocks visible on the user interface.

Figure 7.3: A label confidence value, corresponding to the degree of certainty of the classification, is drawn at the bottom of each text block. The model can therefore be improved by integrating text blocks having low confidence values. Again, text blocks that have been integrated in the document model are surrounded by a thin rectangle. Note that the currently selected text block is slightly highlighted, its label and font size are also displayed.

The dynamic labeling (or classification) of text blocks allows the user to visually estimate the accuracy of the model by simply observing the assigned colors. Additionally, a label confidence value, corresponding to the degree of certainty relative to its classification, is shown as a thin horizontal bar displayed below each text block (see Figure 7.3). Thanks to these visual hints, the user may decide to improve the document model either by labeling some misclassified text blocks (thanks to the contextual popup menu), or by validating a text block having a low confidence rate (simply by clicking on it).

When the user observes that all the text blocks of a page are correctly classified by the system, it can select "integrate page" in the contextual menu in order to automatically add all the text blocks of the current page into the document model. Thus, the model will contain a larger number of samples in its training set, therefore improving its knowledge. Similarly, the user can tell the system to add all the text blocks of the whole document in the model by selecting "integrate document" in the contextual menu.

Once the document model is stable and that the recognition error is deemed to be significantly low, the user can apply it on untrained canonical documents in order to recognize their logical structures, i.e., classify their non-labeled text blocks. Of course, in case of poor recognition, i.e., if some incorrect classifications appear, the document model can still be

improved by re-labeling some misclassified text blocks, thanks to the incremental learning facilities of Dolores.

Finally, logically labeled canonical documents can be stored on disk as OCD+ files, i.e., annotated OCD files. An OCD+ file respects the OCD standard and uses the existing *g* markup element to group graphical primitives together and tag them. This way, high level tree structures can be created without much effort. OCD+ adds three attributes to the *g* markup element: *type*, *name*, and *confidence*, which are used to specify the type of the group element, its name, and a confidence value, respectively. More precisely, a logical label is defined by nesting a text block with a group element having a *type* attribute equals to "label", a *name* attribute equals to the name of the label, and a *confidence* attribute equals to the degree of certainty of the assigned label. Since the basic OCD formalism is left unchanged, i.e., no markup element is added, OCD readers can process OCD+ files transparently.

7.3.2 Document Model

In Dolores, a document model is inferred thanks to the interactions of the user with the interface, based on feedbacks from the dynamic learning system. The user identifies misclassified objects (text blocks) by associating them to classes (logical labels), thus increasing the model knowledge. The model is then used to propose an updated classification of the objects taking into consideration the user's actions.

Such features and functionalities imply that the elaboration of the document model has to take some crucial considerations into account. Indeed, the dynamic interface of Dolores generates a training set of labeled text blocks, therefore statistical models are an ideal solution since they are inferred from training samples. Moreover, the classification process cannot be decoupled from a previous learning process. Thus, both learning and classification processes have to be performed very efficiently because Dolores must ensure a good "reactivity", i.e, the processing time after each action must be reasonably short.

Various learning and classification methods are based on statistical features, and among them, three main techniques corresponding to our needs were retained:

- K-nearest-neighbors or KNN
- Artifical neural networks or ANN
- Support vector machines or SVM

On a first try, we did some preliminary experiments with the k-nearest-neighbors technique (with k=1); the KNN classifier performed well with very simple document classes, a small set of extracted features, and a few number of text block labels. However, increasing the number of labels and extracted features rapidly led to very poor recognition rates. Thus, we found that SVMs and ANNs would certainly give better results. Since we have a long

experience using artificial neural networks, we decided to integrate an ANN as document model into our learning system.

In Dolores, a document model is basically a neural network trained on a set of samples extracted from labeled text blocks, i.e., ground-truthed data. To be more precise, these samples, which are used to feed the ANN, are composed of feature values extracted from text blocks. Therefore, the definition of a set of features to extract is a key issue for the elaboration of our document model; these extracted features are precisely the subject of the next subsection, i.e., Subsection 7.3.3. Concerning the artificial neural network itself, we developed a customized multilayer perceptron specifically dedicated to our issue, which is described in details in a standalone section, i.e., Section 7.4.

Dolores offers a procedure where document models are handled through the user interface thanks to a model panel which provides the following buttons (see Figure 7.1): "Open...", "Save", and "Save As..." are used to open and store document models, "Trainset..." and "Validset..." are used to choose the training and validation sets used by the document model (i.e., the ANN), whereas "Clear" is used to empty the current document model (i.e., reset its training set).

Dolores also specifies a basic XML-based format, called *DLS* (for DoLoreS), used to store document models on disk. An example of such a DLS file is presented in Listing 7.1. Actually, a DLS file does not contain the document model itself; however it holds all the information required to regenerate them from a set of OCD files.

Listing 7.1: A DLS file storing a document model for the e-book class: "ebook.dls".

```
<!-- a DLS file contains all the information required to regenerate a document
     model -->
<model name="ebook">
  <labels>
    <!-- labels are defined by their names and colors (sRGB color space) -->
    <label name="content" color="00 cc 00"/>
    <label name="chapter" color="cc 33 00"/>
    <label name="footer" color="cc cc 00"/>
    <label name="header" color="ff cc 00"/>
    <label name="title" color="00 00 cc"/>
    <label name="author" color="00 99 99"/>
    <label name="garbage" color="99 99 99"/>
  <labels>
  <files>
    <!-- external OCD+ files are referenced, they contain labeled text blocks
         used to regenerate the training samples -->
    <file name="Alices_Adventures_in_Wonderland_NT.ocd+"/>
    <file name="Around_the_World_in_80_Days_NT.ocd+"/>
  </files>
  <regex-set>
    <!-- regex patterns are stored using the Java regex syntax -->
    <regex pattern="\A\d+\s\w+"/>
  </regex-set>
  <box-set>
    <!-- rectangular regions are stored in 72 dpi "x y width height" -->
    <box rectangle="2 2 393 64"/>
```

```
    <box rectangle="2 480 392 58"/>
  </box-set>
</model>
```

A DLS file structure results in the following pattern: the XML root element is called *model* and contains a *name* attribute specifying the name of the document model. A *model* element then contains four children elements: *labels*, *files*, *regex-set*, and *box-set*. The *labels* element lists the set of *label* elements existing in the document model. Each *label* element asserts *name* and *color* attributes used to define both the name and color of the label. Then, the model keeps track of its training data by referencing some OCD+ documents: the *files* element lists a set of *file* elements used to reference OCD+ files thanks to their *name* attributes. Finally, *regex-set* and *box-set* declare some features customized by the user: regular expressions and rectangular regions (which are detailed in the following subsection). Regular expressions are described in the *pattern* attribute of *regex* elements, whereas rectangular regions are described in the *rectangle* attribute of *box* elements.

7.3.3 Extracted Features

Dolores integrates an artificial neural network as document model. Such a statistical document model requires a set of training samples to infer its knowledge. Thus, a set of features used to extract samples from ground-truthed data (i.e., labeled text blocks) has to be defined.

Extracted features can be divided into two categories: self-related features (intrinsic to a text block) and cross-related features (relative to other page elements, i.e., text blocks, images, or the page itself). Self-related features are themselves divided in morphological features, structural features, and textual features, whereas cross-related features are divided in geometrical relations, textual relations, and label relations.

Dolores implements a wide range of features, i.e., morphological, structural, textual, relational, and even some *customizable* and *dynamic* features are extracted from text blocks; all these features are presented and detailed hereafter.

Morphological features

Morphological features are relative to the morphology of a text block. Dolores extracts some features about the bounding box of text blocks (*min-x*, *min-y*, *max-x*, *max-y*, *center-x*, *center-y*, *width*, *height*, *width/height*) and about their text orientation (*text-rotation*).

Structural Features

Structural features are relative to the inner structure of a text block. Dolores extracts features about the font (*font-size*, *font-style*), about text properties (*interline*, *justification*), and about the number of text entities (*number-of-tokens*, *number-of-lines*, *tokens-per-line*). Note that

some features, like the *font-size*, may slightly vary inside a text block; in such a case, the feature value having the maximum number of occurrences is returned.

Textual Features

Textual features are relative to statistical properties of the text inside a text block. Dolores extracts the percentage of various text types (*percentage-of-caps*, *percentage-of-letters*, *percentage-of-numbers*, *percentage-of-symbols*, *percentage-of-spaces*), as well as boolean values indicating whether or not a text block begins with a capital letter or ends with a dot mark (*starts-with-cap*, *ends-with-dot*).

Relational Features

Relational features are relative to inter block properties. Dolores extracts the font sizes of adjacent blocks (*north-font-size*, *south-font-size*), their block widths (*north-block-width*, *south-block-width*), the distance to adjacent images (*north-image-distance*, *south-image-distance*), their widths (*north-image-width*, *south-image-width*), some boolean values indicating whether a text block overlaps another primitive or not (*text-overlap*, *path-overlap*, *image-overlap*), and also some features relative to the entire page (*page-blocks*, *page-lines*, *page-tokens*, *page-number*, *page-even*).

Customized Features

The precited features are the standard ones used by Dolores. However, beside this default set of features, Dolores offers the possibility to define some customized features of two different types: *region* and *regex* features.

A region feature is created by right-clicking on a text block and choosing "Add region feature..." in the contextual menu. A window then just popups and proposes three region policies: text box region, horizontal region, and vertical region (see Figure 7.4). These are simply used to create region features based on the selected text block bounds, and possibly by extending them to fit the pages width or height. The value returned by a region feature is simply the percentage of overlapping between the bounding box of a text block and the rectangular region feature.

Similarly, a user can create a regular expression feature (or regex) by editing a text field using Perl regex syntax (see Figure 7.5). The value returned by a regex feature is the number of regular expression matches found in a text block.

Dynamic Features

Dynamic features are actually boolean features which are dynamically generated during the labeling process. Dolores currently implements the following dynamic features: *font-name*, *font-size*, *font-color*, and *background-color*. So each time the model integrates a text block

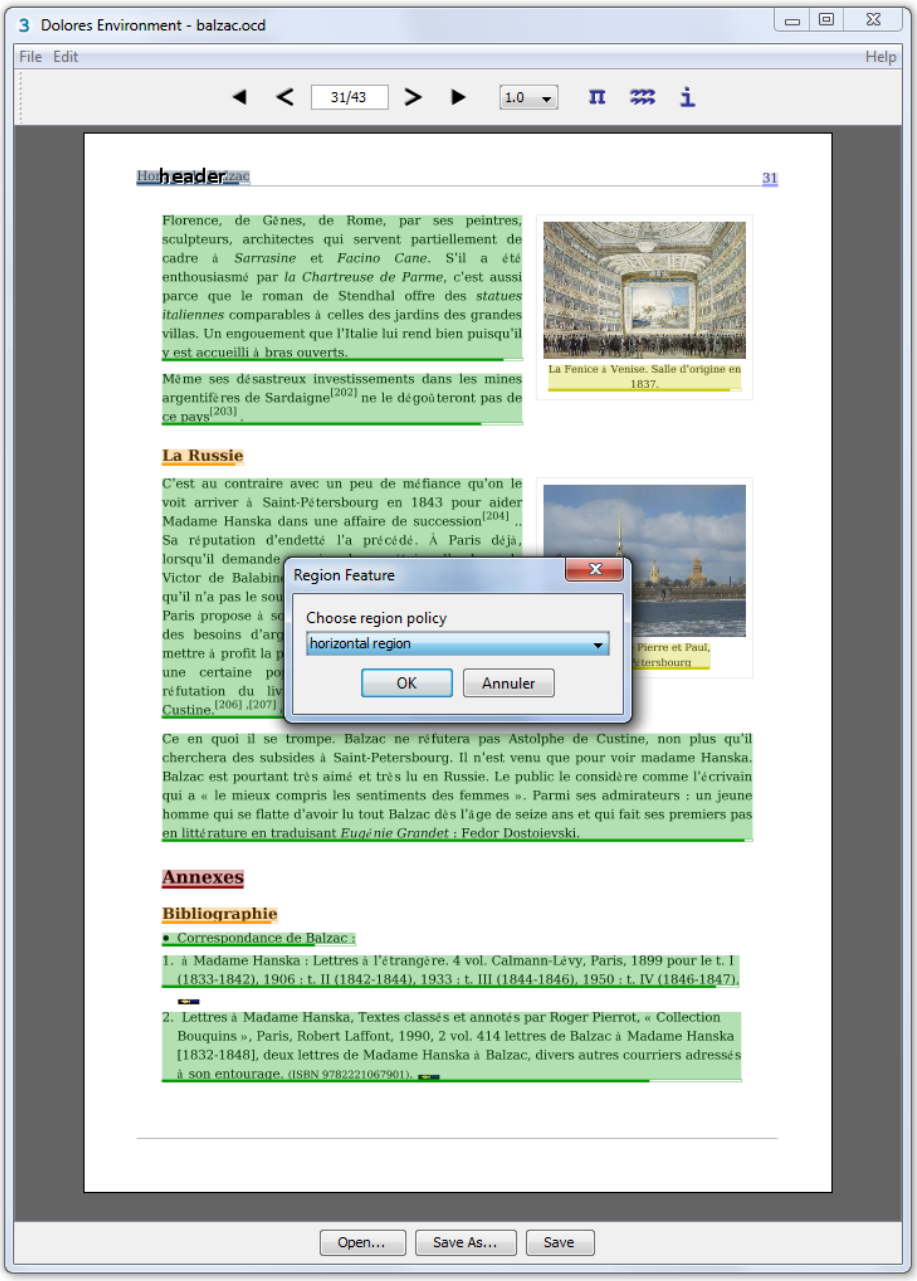


Figure 7.4: The above popup window allows a user to add a region feature to the header text block. The returned value is the percentage of overlapping between a text block and the rectangular region feature. Such a feature is particularly useful when text blocks have recurrent positions, e.g., header, footer, address, copyright, etc.

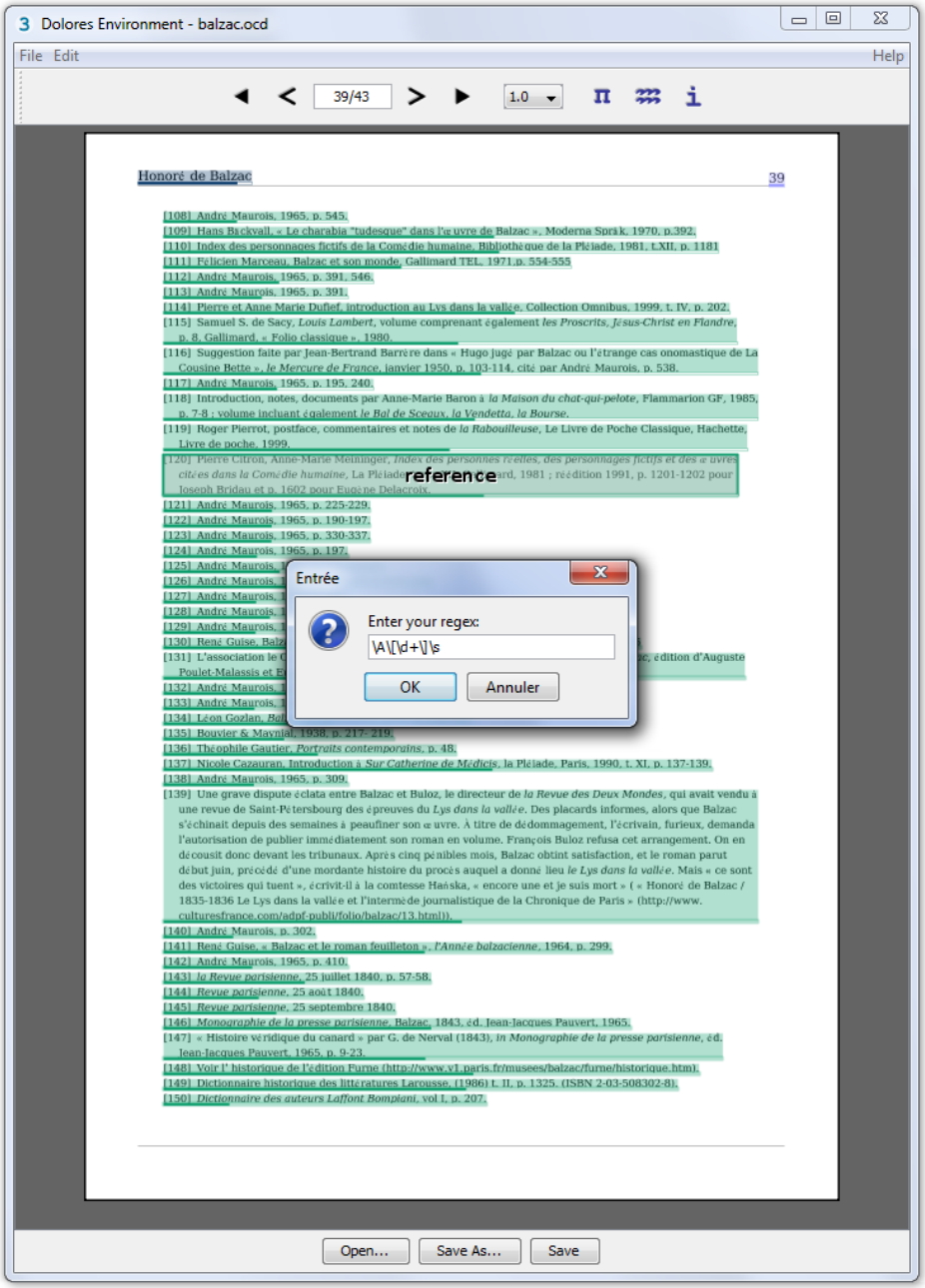


Figure 7.5: The user can define a regular expression feature thanks to a *Perl regex* syntax. The returned value is the total number of regular expression matches found in a text block. Such a feature is especially relevant when text blocks have recurrent textual patterns, e.g., bibliographical references, items from a list, figure captions, etc.

having a new font name, font size, font color, or background color, a corresponding boolean feature is added to the current set of features (e.g., font-arial, font-times, font-12, font-22, font-0000FF, background-AA00FF). Note that *font-size* has already been defined previously. However, the feature is not the same, since the first one returns the value of the font-size, whereas the second one returns 0 or 1, whether or not a text block has a given font size.

7.4 Dynamic Multilayer Perceptron

The document model of Dolores integrates a multilayer perceptron (MLP), because it is a specific type of ANN that is particularly well suited for supervised machine learning. For those who need a little reminder, Appendix B explains in details the concept of multilayer perceptrons. Unfortunately, standard MLPs have some undesirable properties: they are black box systems that need manual configuration and fine tuning to perform efficiently.

Indeed, an MLP is considered as a black box system since its embedded knowledge is not explicit; well, the pure logic of an MLP is clearly defined and understood, however, it is not possible to grasp the meaning underlying the decisions of an MLP. Again, an MLP has a set of parameters that need to be tuned by the user in order to operate properly. In consequence, we developped an ANN directly derived from the MLP which solves the afor-mentionned issues, we call it *dynamic multilayer perceptron* or DMLP.

The aim of a DMLP is therefore twofold: first, facilitating the neural network configuration by removing any manual settings and by using dynamic parameters (Subsection 7.4.1), and second unveiling the black box of standard MLPs by providing dynamic visual feedbacks about the knowledge integrated in the neural links (Subsection 7.4.2)

7.4.1 Architecture and Parameters of DMLPs

A DMLP directly inherits its structure and functionalities from a standard MLP. Additionally, a DMLP sets some constraints about the architecture and parameters of the neural network, which are precisely the topic of the current subsection, i.e., topology, learning rate, activation function, training protocol, stopping criterion, and data normalization. Again, we remind the reader that the theoritecal background about MLPs, and thus DMLPs, is presented and explained in detail in Appendix B; therefore, this subsection does not cover those theoretical aspects.

Topology

The topology of a DMLP imposes strong restrictions to the range of standard MLP topologies. A DMLP has always three layers: an input layer, a hidden layer, and an output layer. The input layer of a DMLP is fully connected to the hidden layer, whereas the hidden layer is only partially connected to the output layer. Actually, an exclusive set of hidden neurons is allocated to each output neuron (see Figure 7.6).

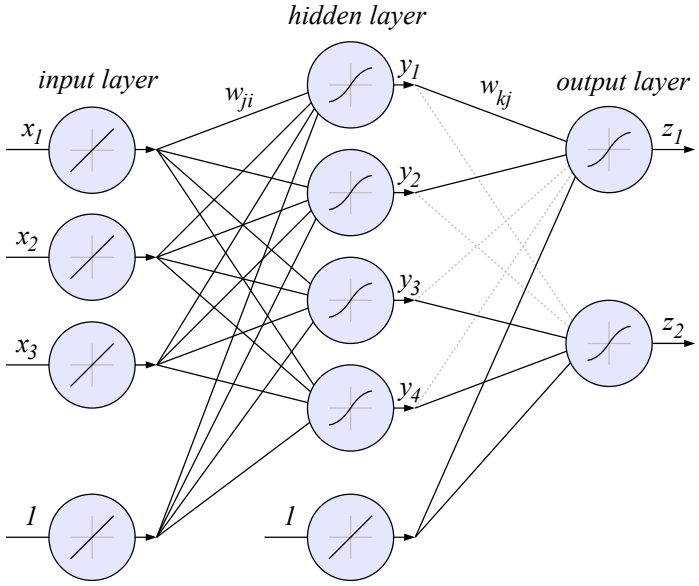


Figure 7.6: A dynamic multilayer perceptron with three input neurons, four hidden neurons, and two output neurons. A DMLP is equivalent to a set of MLPs, each of them dedicated to a unique class. Note that a standard MLP would have fully connected layers, i.e, including the light dashed links in the figure above, whereas a DMLP does not.

Such an architecture allows a unique neural network to behave as a set of distinct MLPs, each of them dedicated to a unique class. This topology enables each output neuron to develop its own decision regions in total freedom, thanks to its dedicated set of hidden neurons, without risking them to be impacted by other classes. Indeed, in standard MLPs, frequent classes tend to overcome the whole system during the training phase. Thus, the total number of hidden neurons of a DMLP is dictated by the number of hidden neurons allocated per output neuron. By experience, we set this number to a default value of three, but of course, it is not hard-coded and may still be customized by an experienced user.

Note that the network topology we propose is the result of many tests achieved during the development of Dolores: various topologies were tested before we reached the conclusion that sets of dedicated hidden neurons performed better than a fully connected hidden-to-output layer. We observed that this topology needed very few learning cycles to achieve its training phase, a real advantage when working with dynamic and reactive environments which require short processing delays in order to be responsive. Moreover, we noticed that the proposed network topology was even able to efficiently deal with quite few training samples, which is also a key issue in an interactive environment where the training data is progressively integrated, i.e., the training set is very small at beginnning.

Data Normalization

In order to achieve properly and efficiently its training phase, an MLP should be fed with input values being more or less comprised in the range [-1.0,1.0] (see Appendix B). A simple technique fulfilling the precited demand is the min-max normalization, which precisely scales and shifts each training sample value in the range [-1.0,1.0]. That is, let x_i be the i^{th} component of a sample, min_i and max_i its minimum and maximum values, then its normalized value $x_i^{'}$ is computed as follows:

$$x_i^{'} = 2 * \frac{x_i - min_i}{max_i - min_i} - 1 \tag{7.1}$$

This normalization technique is appreciable since it is simple, efficient, and ensures that feature values are always spread in the entire input range. For instance, boolean features return -1.0 or 1.0 with the min-max normalization, whereas other normalization techniques (e.g., standard-deviation normalization) do not ensure such a property.

Learning Rate

Each neuron of a DMLP has its own learning rate, which is by default initialized to $\eta = 0.1$. Learning rates then evolve during the network training phase. The training algorithm we use is a modified version of the Quickprop algorithm [31], where the learning rate is related to the second-order function of the error. However, unlike the Quickprop algorithm which uses the error function of a neuron to directly set its learning rate, our algorithm only uses the error function to update its learning rate. In practical terms, after each batch training epoch, i.e., once all the training samples have been presented to the DMLP, the sum-squared-error for each neuron is computed; the learning rate is then lowered if the second-order error function is positive $\eta = \eta - 0.1 * \eta$, and raised if it is negative $\eta = \eta + 0.1 * \eta$.

Activation Function

A DMLP implements a sigmoid function as activation function for hidden and output neurons. Input neurons use the identity function as activation function; indeed, input values are first normalized (see Data Normalization). Figure 7.7 shows the exact sigmoid function implemented in DMLPs (see Equation B.3 in Appendix B) .

Training Protocol

The training protocol used in DMLPs is the batch backpropagation with momentum ($\mu = 0.8$) and weight decay. Indeed, on-line training is not useful since we do not have huge learning sets or memory constraints, and stochastic training is preferable with large redundant training sets, that is not our case (see Appendix B). Moreover, dynamic learning rates used in DMLPs require the computation of the sum-squared-error for each neuron after each batch backpropagation.

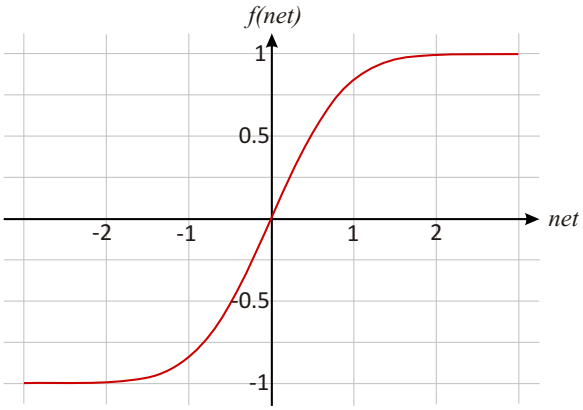


Figure 7.7: A sigmoid function of the form $a \tanh(b\, net)$, where $a = 1.716$ and $b = 2/3$.

Weights Initialization

The weights initialization is a mandatory step to ensure an efficient and uniform learning of the neural network (see details in Appendix B). The weights initialization takes place before the training phase, it is triggered by either an update of the training set or a modification of the number of input features. This way, new samples and added features are instantly fully integrated and considered by the DMLP.

Stopping Criterion

A fundamental issue about the batch backpropagation protocol is the definition of a criterion used to stop the learning phase. Since the training process of a DMLP tend to be entirely automated, a complex stopping criterion would not be tolerable. Thus, as a basic stopping criterion, we simply use the recognition rate over the training set. More precisely, the training process automatically stops after a serie of three batch backpropagation cycles where the overall recognition rate over the training set is equal to one. However, complex datasets may jeopardize a full recognition of the training set and, consequently, the stopping criterion may not be fulfilled. Hence, a customizable maximum number of batch cycles ensures that the training phase stops, and that it is rapid enough in the context of a dynamic and reactive environment.

Let us note that a full recognition during three batch backpropagation cycles is far from a perfect rule, but still, it results from the following considerations: a DMLP should be able to fully recognize its training data while it should avoid their overfitting. Since overfitting is induced by an excessive number of training cycles, the training should stop once all the training samples are recognized. Moreover, three cycles should ensure that the full recognition is not due to some learning "hiccup". Again, our stopping criterion does not fully ensure non-overfitting since complex datasets may still lead to over-training.

7.4.2 Dynamic User Interface of DMLPs

Since the topology and properties of a DMLP are dynamically set, its interface allows only a few parameters to be customized: the number of maximum training cycles and the number of hidden neurons allocated per output neuron. In fact, the interface of a DMLP is, above all, used to observe and get the knowlegde integrated by the network during the training process.

Indeed, a substantial innovation brought by DMLPs is the definition and use of a *relevance criterion* (see Relevance Criterion) that allows a user to visually appreciate the discriminating power of each input feature, either relatively to the whole network or to a given output neuron (see User Interaction and Display Modes). Thus, the knowledge integrated in the network is revealed thanks to the following visual feedbacks: color is used to express neuron activations and link weights whereas size (or thickness) is used to express neuron and link relevance values (see Network Diplaying).

Relevance Criterion

Multilayer perceptrons are considered as "black box" systems since their internal decision mechanism is completely opaque to the user. Actually, a user only observes the inputs and outputs of an MLP while the internal inferred knowledge remains a mystery.

On the opposite, the embedded knowledge of a DMLP is revealed thanks to a relevance criterion through its user interface. The relevance criterion is a value that is computed for each neuron in order to define its influence (or power) relatively to either the whole network, or a given output neuron (i.e., relative to a class). The calculation of each relevance value is based only on the network link weights and, thus, it is a representation of the network knowledge.

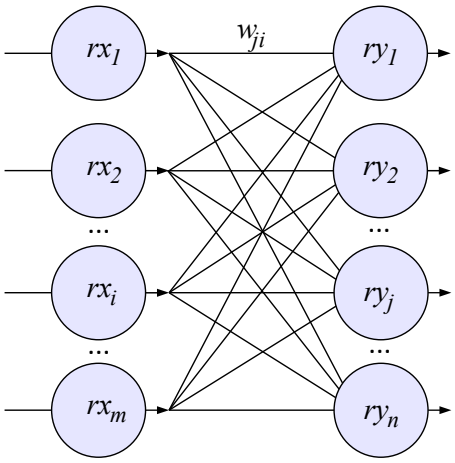


Figure 7.8: The relevance value of a neuron (rx_i) is entirely dependent on its output link weights (w_{ji}) and their corresponding neurons' relevance values (ry_j).

On that account, the relevance value of a given neuron is equal to a normalized weighted sum (using the absolute value of link weights) of its descendant neurons' relevance values. More formally, let rx_i be the relevance value of a neuron on a given layer, ry_j the relevance value of a neuron on the descendant layer which contains n neurons, and w_{ji} the weight of the in between link (see Figure 7.8); the following formula gives the relevance value of a neuron:

$$rx_i = \frac{1}{\sqrt{n}} \sum_{j=1}^{n} |w_{ji}| * ry_j \tag{7.2}$$

The relevance value of each neuron is then computed by back propagating the relevance values from the output layer up to the input layer. A default relevance value equal to 1.0 is set to each neuron of the output layer. The choice of a unitary output relevance value is straightforward since it is primarily used to appreciate the influence of some input neurons on a selected output neuron.

Finally, we also use the term "relevance value" with network links, i.e., the relevance value of a link is equal to the absolute value of its weight multiplied by the relevance value of its output neuron.

User Interaction and Display Modes

The visualization of the information integrated in a DMLP is supported by two display modes; neurons and links are then displayed relatively to the current display mode:

- in *sample mode*, the network is fed with the selected sample and the relevance values are computed according to the entire set of output classes.
- in *class mode*, the network is fed with the average training sample of the selected class and the relevance values are computed according to the selected class (i.e., output neuron).

By default, the interface of a DMLP is in sample mode and thus the network properties are displayed relatively to the selected text block. So, each time the user points to another text block in the labeling panel of Dolores, its feature values are fed forward into the network and the result of the process is instantly revealed by the output layer (see Figure 7.9), i.e., the neuron having the greatest output activation is the "winning" class.

The user can also interactively switch to class mode by pointing the mouse cursor to an output neuron (i.e., label class). At this point, the DMLP interface is instantly updated relatively to the selected output class. This means that only the neurons and links relative to the selected class are taken into account. Moreover, the network is fed with the mean feature vector of the training samples belonging to the selected label class.

Finally, the DMLP interface even allows the user to work with a standard MLP. Indeed, the toolbar of the artificial neural network has some parameters which are disabled by default. However, by setting the number of hidden neurons allocated per output neuron to zero, an

Figure 7.9: Dolores integrates a DMLP as document model. This figure shows an "e-book" document model with eight output classes: "title", "author", "advice", "header", "content", "footer", "part", and "chapter". Note that the network assigns a "footer" label to the input sample, i.e., the text block currently selected in the labeling panel.

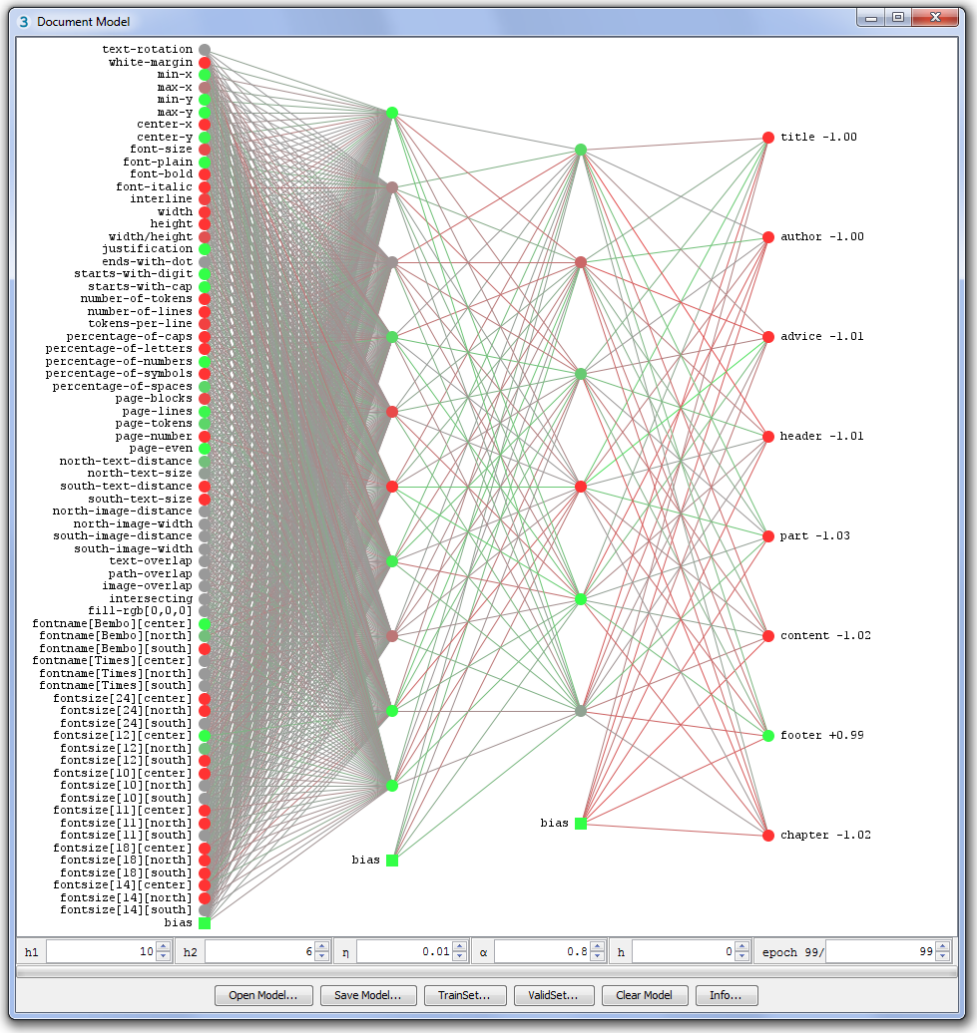


Figure 7.10: Standard MLPs can also be used with Dolores. A dedicated toolbar allows the user to configure the set of parameters of an MLP.

extended set of parameters, corresponding to a multilayer perceptron, are activated (see Figure 7.10).

Network Displaying

As stated earlier, neurons and links of a DMLP are displayed using two visual properties: color and size/thickness. A color scheme is used to express both the output activation of a neuron and the weight of a neural link, whereas a size/thickness property is used to express their relevance values (see Figure 7.9).

More precisely, we defined a simple color scheme allowing the user to visually appreciate link weights and neuron output activations: red represents a value of -1.0 or below, green represents a value of 1.0 or above, and gray represents a value of 0.0. An intermediate value is then represented using a linear combination of red and gray if the value is negative, or green and gray if the value is positive.

The size (i.e., radius) of a neuron as well as the thickness of a link are both proportional to their relevance value, which expresses the sollicitation degree of a neuron or link according to either a given output class or the whole set of output classes (i.e., given the display mode). Additionally to thickness, neural links are drawn using alpha values, i.e., links having high relevance values are completely opaque whereas links having low relevance values tend to be transparent. Thickness and alpha value together provide some precious visual information as they emphasize strong links relatively to weak ones, therefore clarifying the perception of the whole network by hiding neural links having an insignificant impact on it.

Another precious visual clue is given by the ordering of the input neurons. Indeed, a dynamic ranking ensures that they are always sorted according to their relevance values, i.e., the most relevant neurons at the top. Such a ranking allows a user to easily pick out the most significant input features. The ranking reveals the set of features that have the strongest influence on the decision function either for the entire set of classes or given a selected class. For instance, Figure 7.11 shows a DMLP in class mode: we observe that only neurons and links related to the selected class are taken into account and, thus, the ranking of the input features is relative to the “content” label class.

Finally, the relevance criterion is also used to trim the number of input neurons displayed on the interface. Indeed, since input features are dynamic, their number may become quite large. Therefore, input neurons having a relevance value below a given threshold (0.01 by default) are left aside.

7.5 Conclusion

This section presented Dolores, an elaborated environment that aims at producing document models thanks to interactive and incremental learning facilities. The innovation of Dolores partly resides in its interactive and dynamic environment that greatly facilitates and speeds

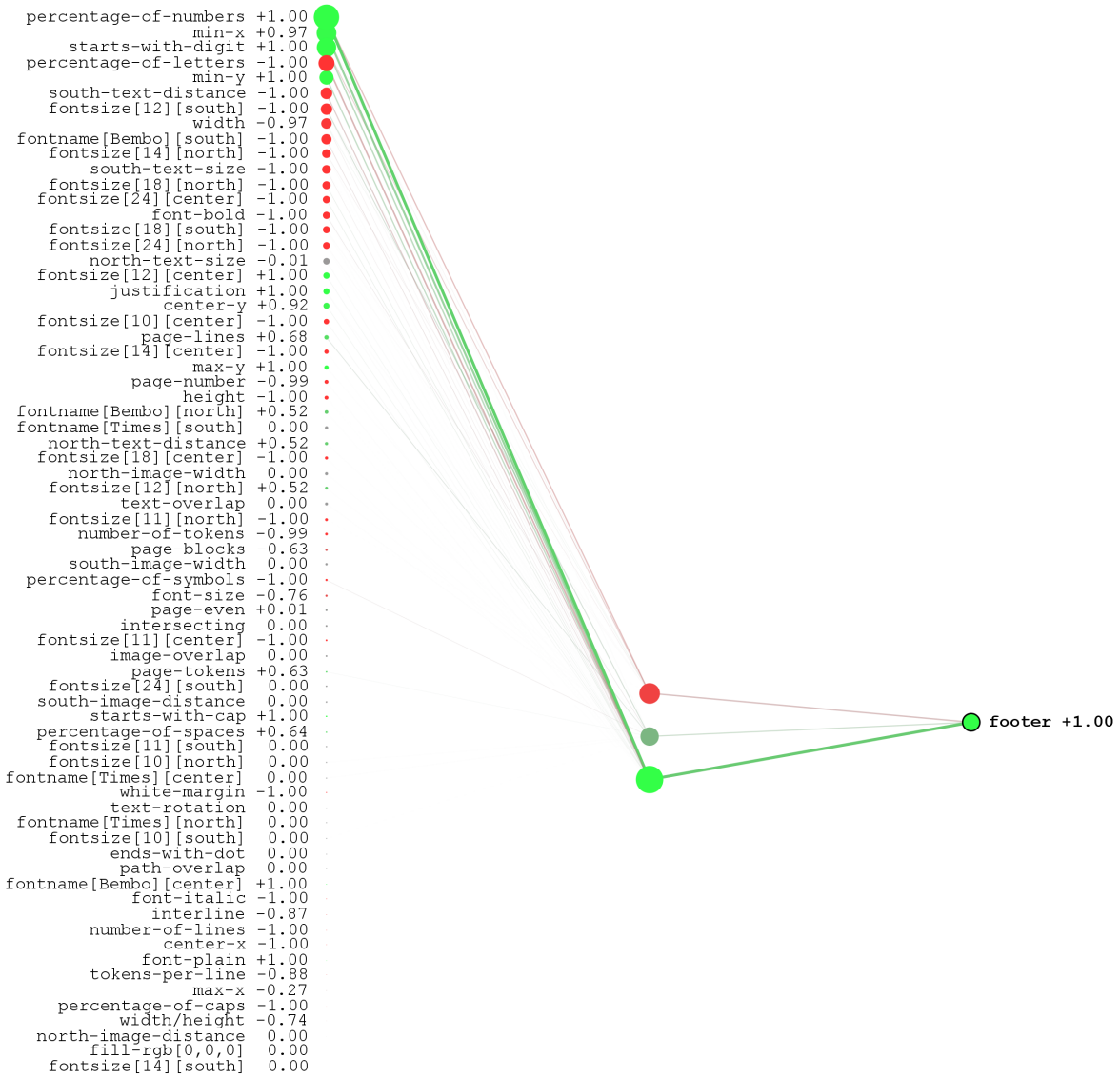


Figure 7.11: The input features of a DMLP are sorted according to their relevance values. By pointing the mouse to an output neuron, the user can visualize the most relevant features relatively to the selected label class.

up the creation of document models. Henceforth, a user infers a model by iteratively labeling text blocks and observing the classification improvements on a document. Thus, after each user action, the learning system of Dolores dynamically updates its document model; the knowledge freshly acquired is then instantly reflected through the interface.

Another great innovation of Dolores is held by its dynamic multilayer perceptron. Indeed, the document model of Dolores is backed by a DMLP, a powerful neural network derived from standard MLPs that does not need any manual configuration. Note that performance comparison between MLPs and DMLPs have not been explicitly reported into this dissertation since MLPs are simply unable to perform efficiently (or even to cope) with training sets containing strong unbalanced label class sets (i.e., label classes having totally different a priori probability). Furthermore, contrary to MLP, the interface of a DMLP offers a visual representation of the knowledge learned during the training phase. Thanks to the definition of a relevance criterion itself used in conjunction with a dynamic ranking of the input neurons and adequate visual clues (color and size), the interface of a DLMP allows a user to quickly grasp the most relevant features used by the network to discriminate its output label classes. Therefore, the graphical interface of a DMLP unveils the black box of standard neural networks by revealing its embedded knowledge; such a "transparent box" is a real break-through compared to standard MLPs.

Nevertheless, several aspects of the DMLP may still be improved in the future. For instance, the number of hidden neurons dedicated per output neuron is by default set to three, a value that has been experimentally set. However, it is clear that a more objective and dynamic way of determining this number should be investigated. Similarly, the training stopping criterion is a bit subjective since it looks for a full recognition of the training data during three iterations. This criterion is subject to discussion and alternatives should be studied too (e.g., by looking at the error on a validation set). Beside this, the relevance criterion could be used to prune the network topology by removing neurons and links having very low relevance values, i.e., having a negligible impact on the network.

Concerning the set of features extracted from each text block, it is (and will always be) subject to extension, e.g., textual relations and label relations are missing. However, labels are precisely the outputs of the neural network, as a consequence, label relations should be modeled by reinjecting network outputs to inputs; such a topology would imply a recursivity in the network which should improve significantly the recognition rate at every recurrent cycle.

A current limitation of Dolores relates to its document model which is uniquely based on the labeling of text blocks without taking their interrelations into account. Thus, the logical structure is represented as a flat structure of labels, which is quite restrictive especially in the case of complex documents having deep logical structures. The recognition of the hierarchical structure would certainly be a great challenge to tackle in the near future. A possible solution would be to interactively infer grammatical rules keeping track of the logical

hierarchy thanks to small deterministic automatons. Such rules would simply be integrated in the existing document model of Dolores. As a concrete example, the logical hierarchy of a scientific paper may be described by the following set of rules: a scientific paper begins by a title, followed by the author names, the abstract, the sections, and subsections themselves containing the content of the article which can be composed of text blocks, images, and graphics. Although this example is basic, deep and complex logical structures could also be described with such a mechanism. Of course, recovering this structure is straightforward for simple document layouts, i.e., having one or two columns, but it is definitely more complicated with documents having complex layouts and deep logical structures.

To put it in a nutshell, this chapter presented Dolores, an incremental learning system able to recover the logical structure of electronic documents by interactively inferring their document models. So far, we exposed in details the internal mechanisms as well as the user interface of our innovative learning system; therefore, next chapter, i.e., Chapter 8, focuses on some concrete applications of Dolores and reports on a set of evaluations we did about representative case studies.

8

Case Studies

Contents

This chapter assesses the usability and efficiency of Dolores through the evaluation of a set of concrete experiments. The goal of this chapter is twofold: first, to validate the canonical document format as a physical and pivot document format for high level processing, and second, to demonstrate the relevance and efficiency of Dolores as a system for the logical restructuring of documents.

However, the evaluation of such a system requires an accurate methodology. Therefore, the first section of this chapter, Section 8.1, precisely defines a protocol scheme used to evaluate our dynamic and interactive learning system. Then, Section 8.2, 8.3, 8.4, 8.5, 8.6, and 8.7 present a set of representative case studies of document logical structure recognition thanks to various document classes going from very basic to quite complex layouts. Section 8.8 demonstrates that Dolores is also able to identify the class of a document according to a predefined set of document models. Finally, Section 8.9 concludes the chapter.

8.1 Evaluation Protocol

The evaluation of a learning system is an important issue, requiring the definition of a sequence of tasks, themselves relative to a set of criteria, in order to quantify its performance,

i.e., the evaluation protocol. For instance, off-line learning systems usually evaluate their recognition performance thanks to batch processes performing cross-validation on ground-truthed data. Dolores can't apply such an evaluation protocol, since it is based on a non standard approach using a dynamic and interactive learning system. Thereby, the evaluation protocol of Dolores should precisely quantify these dynamicity and interactivity features during the learning process, i.e., the document model generation. The evaluation protocol of Dolores also highlights the performance of inferred document models when applied to new documents.

Following the aforementionned considerations, a specific evaluation protocol has been elaborated. In order to get a relatively good appreciation of a document model and its generation, we divide the evaluation protocol into two distinct phases: a first one used to infer the document model, and second one dedicated to its evaluation. For this purpose, we use a set of five documents per evaluation; the first three documents are used to infer the document model, whereas the two last ones are only used to appreciate its performance.

The evaluation protocol adopts the following procedure: the first document is entirely integrated in the model through a dynamic and interactive labeling; the recognition rate is computed after each user action and reported on a graphic showing the evolution of the model relatively to the number of labeled objects, i.e., user actions. This basic model is then applied to the second document, the recognition rate is again reported, however, only the misclassified text blocks are labeled by the user and integrated in the document model, i.e., the document model is incremented. The same process is applied to the third document. Finally, the fourth and fifth documents are only used to observe the performance of the document model by looking at the classification matrices and recognition rates.

8.2 Case Study 1a: TV Schedules, Hard-Coded

For the first case study, we want to show that canonical document formalism is really helpful to quickly recover the logical structures from documents having a "not so complex" layout. Thus, we elaborated a basic hard-coded prototype able to recover the logical information from TV schedules. Note that this hard-coded prototype does not benefit from the interactive environment of Dolores; indeed, it is implemented from scratch and is only based on the canonical document formalism.

Foremost, we conceived a simple DTD corresponding to our requirements (see Listing 8.1). We then downloaded TV schedules (6 different TV channels) from the Swiss TV website during one week (see Figure 8.1). 42 PDF documents were generated from these HTML TV schedules (6 channels during 7 days), processed through XED and transformed into 42 OCD files.

Listing 8.1: We elaborated the following DTD as a template used to generate the logical XML files containing the TV schedules.

```
<!ELEMENT tvprogram (tvdate+)>
<!ELEMENT tvdate tvchannel+)>
<!ELEMENT tvchannel (tvshow+)>
<!ATTLIST tvprogram description CDATA #REQUIRED>
<!ATTLIST tvdate year CDATA #REQUIRED>
<!ATTLIST tvdate month CDATA #REQUIRED>
<!ATTLIST tvdata day CDATA #REQUIRED>
<!ATTLIST tvchannel channel CDATA #REQUIRED>
<!ATTLIST tvshow time CDATA #REQUIRED>
<!ATTLIST tvshow title CDATA #REQUIRED>
<!ATTLIST tvshow description CDATAT #REQUIRED>
```

Based on the previous 42 canonical documents extracted by XED, we implemented an hard-coded analysis tool able to recover their logical structure in three simple steps:

1. Since TV show times are always written in a rigorous syntax, they can be expressed and recognized thanks to the following regular expression: `\A\d\d:\d\d\z`, in Perl style ("`\A`" means "beginning of the text input", "`\d`" means "one digit", and "`\z`" means "end of the text input").

2. TV show descriptions are then recognized by locating the closest text blocks at the right of each TV show time.

3. TV show titles and descriptions are finally recognized by looking at the font face of each token, i.e., bold for title and plain for description.

Listing 8.2: An extract of the resulting XML logical file containing the detailed TV schedule is presented below.

```
<tvprogram descripiton="TSR schedule">
 <tvdate year="2005" month="09" day="25">
  <tvchannel channel="TSR1">
   <tvshow time="07:00" title="Quel temps fait-il ?" description=""/>
   <tvshow time="09:10" title="Le doc expeditions" description="A la poursuite
        des pierres precieuses"/>
   <tvshow time="10:00" title="Dieu sait quoi" description="Le couple : beni
       ou honni ?"/>
   <tvshow time="11:05" title="C'est tous les jours dimanche" description=""/>
   <tvshow time="12:20" title="Racines" description="Le gospel du pasteur"/>
   <!--and so on-->
  </tvchannel>
</tvprogram>
```

As a result, we obtained the XML structure presented in Listing 8.2, corresponding to the previous DTD. The XML logical file generated from all the 42 canonical files was free of errors. These accurate results were, of course, trivial to extract since we dealt with quite basic layouts and logical structures. However, this example proved that it was possible to implement an easy and intuitive hard-coded algorithm in a relative short time by using our canonical document representation.

Figure 8.1: A PDF document generated from the website of the Swiss TV schedule from September 25, 2005.

8.3 Case Study 1b: TV Schedules, Dolores

After implementing an hard-coded prototype, we tried to recover the logical structure of TV schedules by interactively inferring a document model with Dolores, our interactive learning environment for the assisted acquisition of document models.

Thus, we interactively labeled the first OCD document with the following labels: "time", "show", and "garbage". Figure 8.2 shows that the labeling of only 11 objects in this first document inferred a model able to achieve a 100% recognition rate. The second and third documents were then perfectly classified using this freshly inferred document model, thereby, we left it unchanged, i.e., we did not increment it. Finally, we applied our model on the fourth, fifth, and even all the remaining TV schedules OCD files and achieved a perfect recognition of all text blocks.

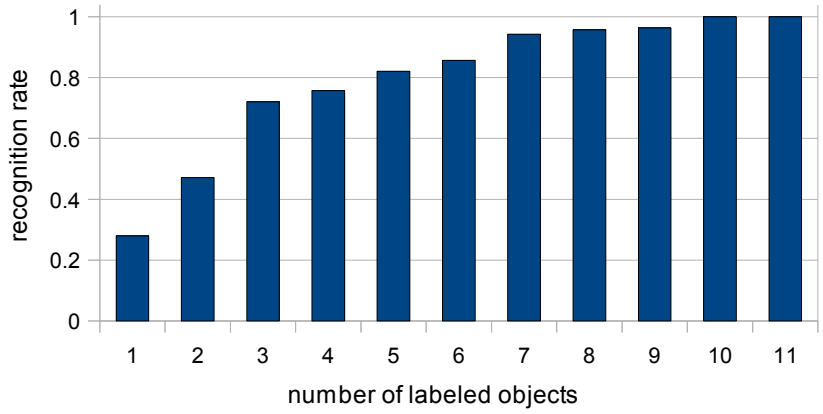


Figure 8.2: The interactive labeling of the TV schedules is very fast. All text blocks of the first document are recognized after only 11 actions of the user.

The efficiency of the labeling comes from the dynamic interface of Dolores (see Figure 8.3), which reflects instantly the integrated knowledge by classifying on the fly the visible text blocks. This visual feedback holds the advantage of labeling only some misclassified text blocks; if all the text blocks of a page are correctly classified, the whole page can directly be integrated in the document model (or even the whole document).

8.4 Case Study 2: E-books

As a second experiment, we chose a class of documents having a more complex logical structure while keeping a basic layout. On that account, we downloaded several PDF e-books from the Planet PDF database [85]. Among them, we retained five big e-books composed of parts and chapters (sorted by author's name): "Nostromo: A Tale of the Seaboard" by Joseph Conrad (790 pages, 4'307 text blocks), "Crime and Punishment" by Fyodor Dostoevsky (967

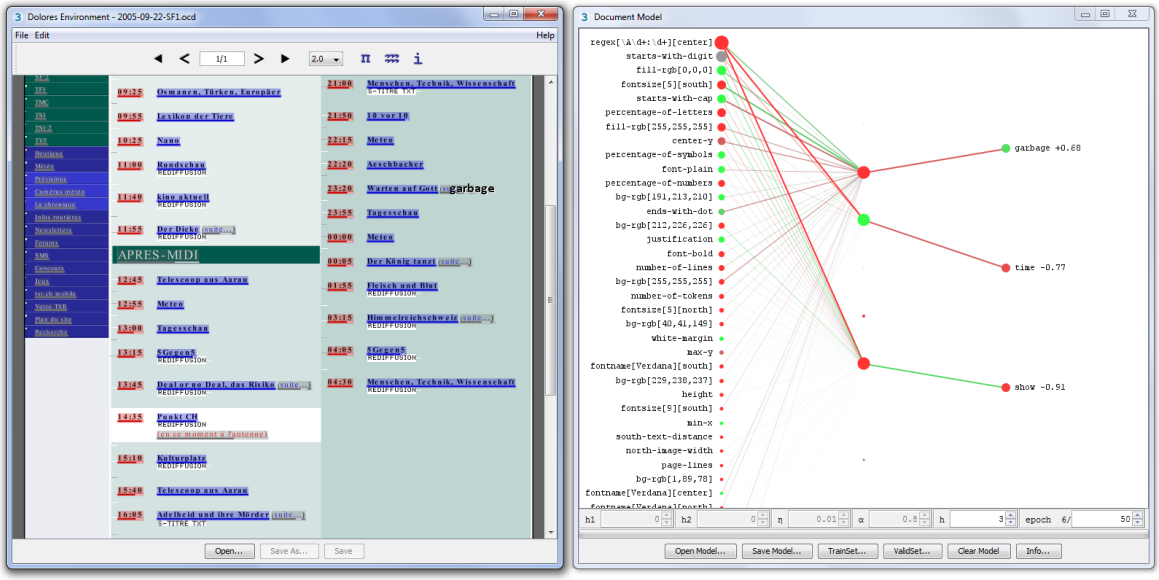


Figure 8.3: The interactive and dynamic labeling interface of Dolores at work with the TV schedules case study: on the left, the labeling interface itself, on the right, the dynamic multilayer perceptron (i.e., the document model).

pages, 6'056 text blocks), "Madame Bovary" by Gustave Flaubert (570 pages, 3'936 text blocks), "Sons and Lovers" by D. H. Lawrence (801 pages, 7'143 text blocks), and "Anna Karenina" by Leo Tolstoy (1759 pages, 11'768 text blocks).

Following our evaluation protocol, we labeled the first e-book "Nostromo: A Tale of the Seaboard". Figure 8.4 shows the evolution of the recognition rate relatively to the number of labeled objects, i.e., text block, page, or document. One can observe that 13 labeling were enough to achieve a full recognition of this first e-book. The second e-book, "Crime and Punishment", was then classified thanks to the freshly inferred document model and resulted in a recognition rate of 99,98%. The classificiation matrix of Table 8.1 shows that there was only one misclassified text block, i.e., an "author" text block classified as a "content" one. Thanks to the incremental learning facilities of Dolores, we simply labeled this misclassified text block in order to increment the existing document model. This updated model was then successfully applied on "Madame Bovary" since we got a recognition rate of 100%. The recognition was perfect, thereby, we left the document model unchanged (no need to apply any incremental learning). The model was finally applied on "Sons and Lovers" and "Anna Karenina" with, again, a recognition rate of 100% in both cases.

The e-book document class has a simple layout, with a low intra-class variability. Therefore, the generation of a document model is easy, quite fast, and results in a perfect recognition rate on untrained documents. As presented in the previous chapter, i.e., Chapter 7, the document model of Dolores is based on a dynamic multilayer perceptron, a learning system

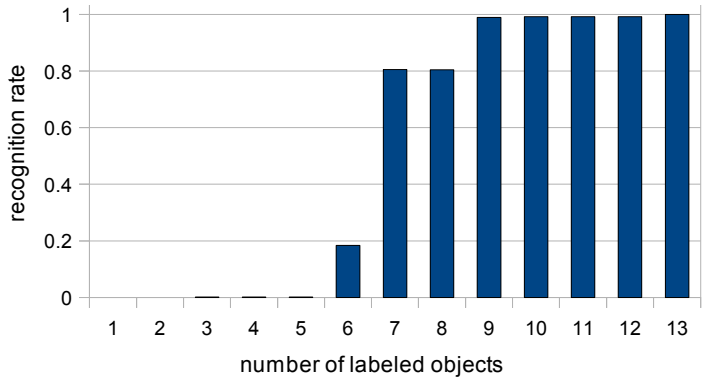


Figure 8.4: The interactive labeling of the e-book "Nostromo: A Tale of the Seaboard" is straightforward. Text blocks are all recognized after 13 labelizations.

Label	Predicted x Actual Label								Total
title	1	0	0	0	0	0	0	0	1
author	0	1	0	0	1	0	0	0	2
advice	0	0	1	0	0	0	0	0	1
header	0	0	0	966	0	0	0	0	966
content	0	0	0	0	4071	0	0	0	4071
footer	0	0	0	0	0	966	0	0	966
part	0	0	0	0	0	0	8	0	8
chapter	0	0	0	0	0	0	0	41	41

Table 8.1: Classification matrix of "Crime and Punishment" using the model generated with "Nostromo: A Tale of the Seaboard". Only one text block is misclassified, i.e., an "author" text block is classified as "content".

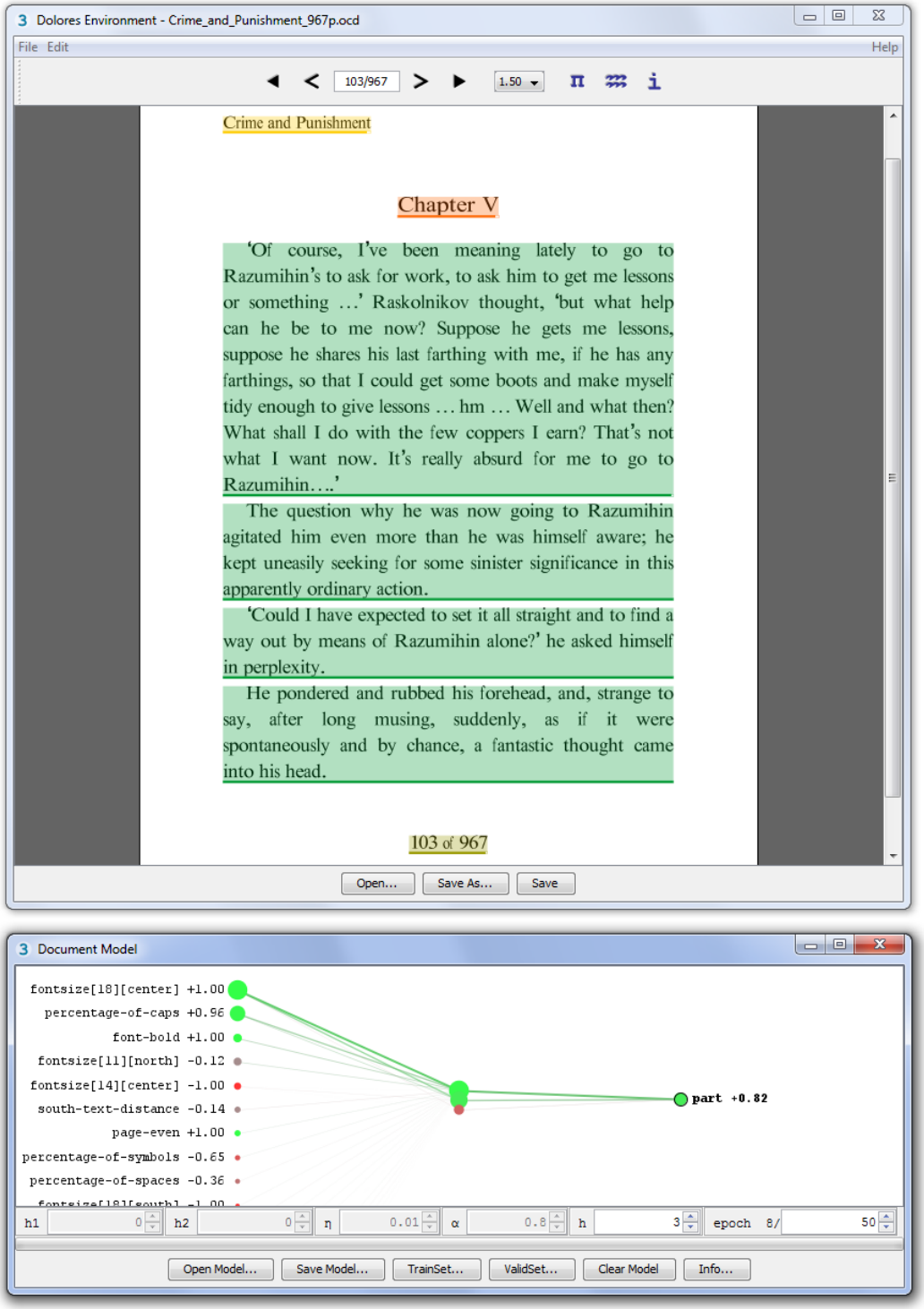


Figure 8.5: The labeling window is displayed on the top of the figure, whereas the model window is displayed on the bottom. The e-book DMLP is showing its most pertinent features relatively to the "part" label (which is not present on the currently displayed page). The three most pertinent features used to identify a "part" are the font size, the percentage of capitals, and the boldness of a text block.

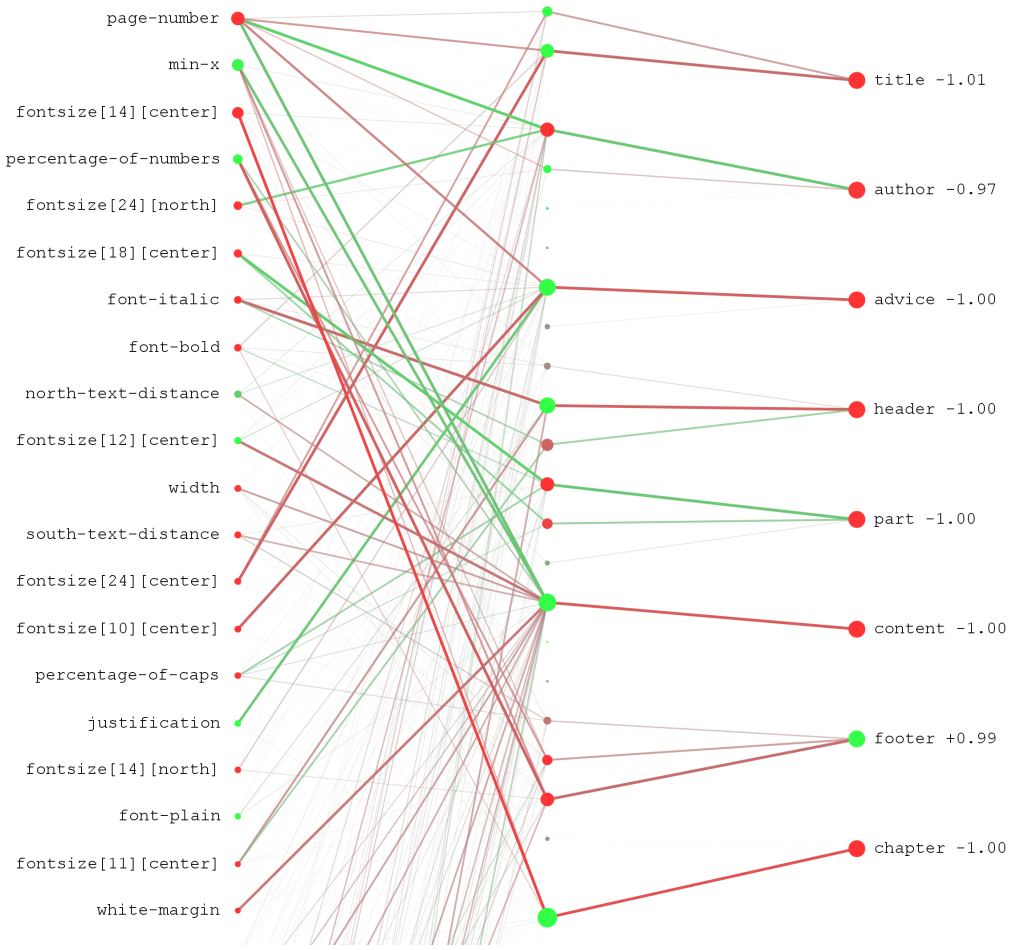


Figure 8.6: A dynamic multilayer perceptron trained with one complete e-book, "Nostromo: A Tale of the Seaboard", and incremented with two others, "Crime and Punishment" and "Madame Bovary".

allowing the user to visually appreciate the relevant features used to discriminate each label class. Figures 8.5 precisely reveals the relevant features relatively to the "content" label, whereas Figure 8.6 shows the whole dynamic multilayer perceptron. Surprisingly, the feature that most discriminates the "content" label is the percentage of number in a text block. Then, comes the font size, followed by the margin feature, i.e., the size of the white space margin surrounding a text block.

8.5 Case Study 3: Wikipedia Articles

For the third experiment, we downloaded five PDF versions of Wikipedia articles on famous French writers: "Louis Aragon" (11 pages, 175 text blocks), "Honore de Balzac" (43 pages, 767 text blocks), "Charles Baudelaire" (11 pages, 203 text blocks), "André Breton" (16 pages, 295 text blocks), and "Albert Camus" (10 pages, 182 text blocks).

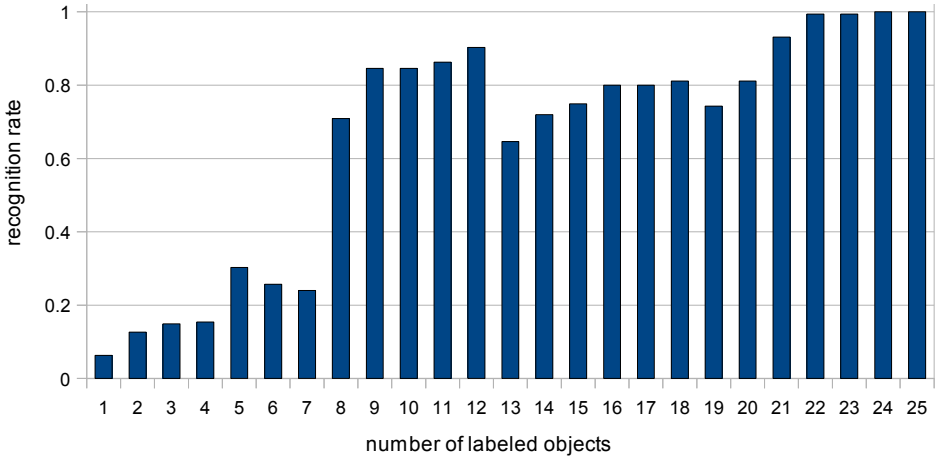


Figure 8.7: The complete integration and recognition of the first Wikipedia article, "Louis Aragon", in the document model is achieved after the labeling of 25 objects.

A first version of the model was inferred by labeling the whole article "Louis Aragon". Figure 8.7 shows the recognition rate of this first document relatively to the number of labeled objects. This first version of the model was then applied on the second Wikipedia article, "Honoré de Balzac", with a recognition rate of 87.7%. Indeed, Table 8.2 shows that many "reference" text blocks are misclassified as "content". Since "reference" and "content" have very close layouts, we decided to add a regular expression feature matching the start of a reference in order to improve the recognition results of the DMLP.

Once the regular expression integrated in the system (i.e., "`\A[\d+]\s`" in Perl regex), we incremented the document model by labeling the remaining misclassified text blocks. The resulting document model was then applied on the third Wikipedia article "Charles

Label	Predicted x Actual Label										Total
header	43	0	0	0	0	0	0	0	0	0	43
page-nb	0	43	0	0	0	0	0	0	0	0	43
title	0	0	4	0	0	0	0	0	0	0	4
intertitle	0	0	0	2	0	0	0	0	1	0	3
content	0	0	0	0	355	0	0	0	3	0	358
section	0	0	0	0	0	9	0	0	0	0	9
caption	0	0	0	0	1	0	35	0	0	0	36
subsection	0	0	0	0	0	0	0	38	0	0	38
subsubsection	0	0	0	0	0	0	0	0	6	0	6
reference	0	0	0	0	89	0	0	0	0	138	227

Table 8.2: Classification matrix of "Honoré de Balzac" using the model generated with "Louis Aragon". Many "reference" text blocks are misclassified as "content" ones; as a consequence, the recognition rate reaches its limit at 87.7%.

Baudelaire", with a recognition rate of 100% (and thus, no need to increment the model). Finally, we applied the model on the two remaining articles, i.e., "André Breton" and "Albert Camus", both of them were also perfectly recognized.

Figure 8.8 displays the pertinent features used by our model relatively to the "caption" output label. As one can expect, we observe that the "caption" label is mostly discriminated by the distance between a text block and its above image.

8.6 Case Study 4: Scientific Papers

For the fourth experiment, we chose a class of documents having a more complex layout: two columns scientific papers. We actually downloaded five scientific papers being part of the "Computer" magazine of January 2009 (vol. 42, issue No. 1): "The Changing Paradigm of Data-Intensive Computing" (9 pages, 192 text blocks), "The Organization and Management of Grid Infrastructures" (11 pages, 244 text blocks), "Computing with Proteins" (10 pages, 407 text blocks), "Body Area Sensor Networks: Challenges and Opportunities" (8 pages, 173 text blocks), and "The 24-Hour Knowledge Factory: Can It Replace the Graveyard Shift ?" (8 pages, 148 text blocks).

Prior to the labeling phase, we added three regular expressions, looking for patterns identifying references, figures, and tables. The first scientific paper "The Changing Paradigm of Data-Intensive Computing" was then used to infer a basic version of the document model. The learning evolution of this document is reflected on the Figure 8.9, where a full recognition is achieved after 32 labeling actions. This document model was then applied on the second scientific paper, i.e., "The Organization and Management of Grid Infrastructures". The resulting classification matrix is presented in Table 8.3.

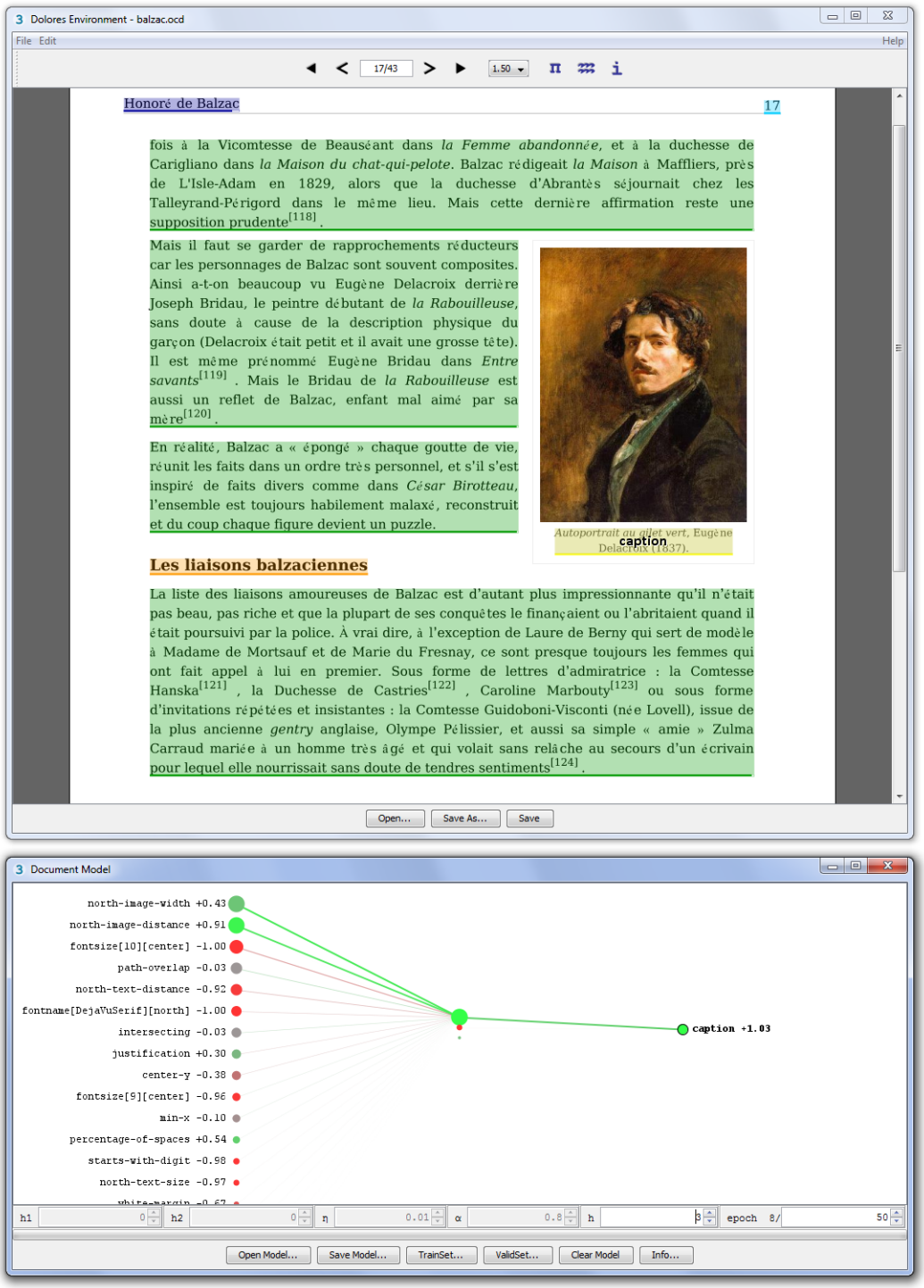


Figure 8.8: Dolores trained with some Wikipedia articles. The most pertinent features relatively to the "caption" label are displayed in the model frame. The three most relevant features are the size of the closest upper image, its distance, and the font size of the text block.

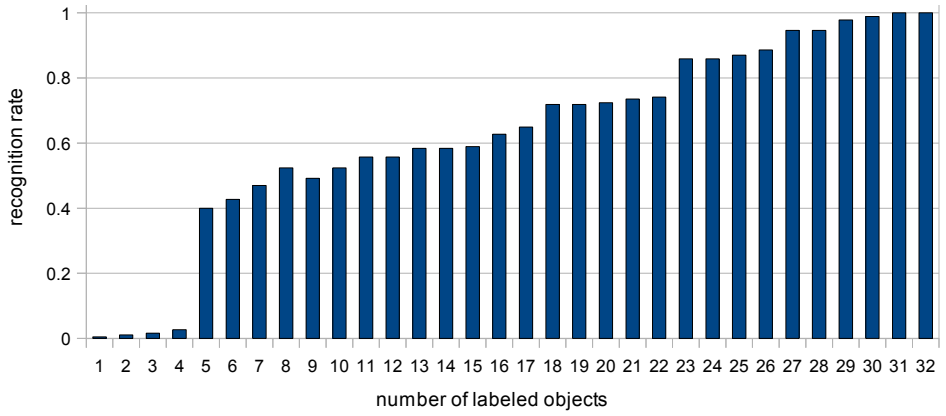


Figure 8.9: The complete integration and recognition of the first paper "The Changing Paradigm of Data-Intensive Computing" in the document model is achieved after the labeling of 32 objects.

In this table, we observe that a lot of text blocks are misclassified, thus leading to a poor recognition rate of 66%. This is explained by the fact that a lot of "content" text blocks are classified as "garbage". Indeed, the "garbage" label gathers both advertisements and other waste text blocks (e.g., text blocks overlaying graphics or images in figures). Unfortunately, the second scientific paper introduces a new layout that emphasizes parts of its content by using new background colors, frames, font faces, and sizes. Since the document model has never encountered such a layout, the DMLP classifies these new text blocks in the most similar label class, which is unfortunately "garbage". Consequently, some of these misclassified text blocks are interactively corrected (i.e., labeled) by the user and thus integrated in the document model (the model is updated or incremented), in order to improve its recognition rate for future classifications. Note that only a few misclassified text blocks have to be labeled because each user action triggers an update of the model, generally improving the classification of the whole set of text blocks.

The updated document model is again applied on the next untrained document, "Computing with Proteins". The obtained recognition rate is now equal to 96.8%, a quite good result since only six text blocks are misclassified. Moreover, all misclassified text blocks are "garbage" text blocks that have been misclassified in other label classes. The document model is once again incremented by labeling some of these misclassified text blocks.

Finally, the remaining two scientific papers are classified using the incremented document model, the recognition rates are equal to 99.4% and 98.6%, respectively. As a matter of fact, a "garbage" text block is classified as "content" in the fourth scientific paper "Body Area Sensor Networks: Challenges and Opportunities", whereas two "garbage" text blocks are classified as "content" and "footer" in the fifth scientific paper "The 24-Hour Knowledge Factory: Can It Replace the Graveyard Shift ?".

Label	Predicted x Actual Label																Total
header	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6
title	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
author	0	0	0	1	0	0	0	0	0	0	3	0	0	0	0	1	5
abstract	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
content	0	0	0	0	81	0	0	0	0	0	58	0	0	0	0	0	139
lettrine	0	0	0	0	0	2	0	0	0	0	4	0	0	0	0	0	6
section	2	0	0	0	0	0	5	0	0	0	2	0	0	0	0	0	9
page-nb	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	11
footer	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	13
intercontent	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	3
garbage	0	0	0	0	5	0	0	0	7	0	16	0	0	0	0	0	28
figure	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	4
subsection	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	7
table	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
reference	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	8
biography	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3

Table 8.3: Classification matrix of "The Organization and Management of Grid Infrastructures" using the model generated with "The Changing Paradigm of Data-Intensive Computing". Many text blocks are misclassified due to the fact that the model has never encountered some layout particularities, the recognition rate is therefore quite poor; it only reaches 66%.

Figure 8.10 shows the interface of Dolores displaying a scientific paper on its labeling panel and showing its most pertinent features relatively to the "section" label on its model panel.

8.7 Case Study 5: Newspapers

As final experiment, we focused on the newspaper class because of its rich layout as well as its detailed logical hierarchy. Thus, we downloaded five consecutive electronic versions of the Swiss newspaper "La Liberté", from the 5th to the 9th of January 2009. The newspaper "La Liberté" is composed of many different parts including advertisements, funeral, and stock exchange pages. Therefore, we extracted a well defined subset of pages from each newspaper, comprising both the International and National parts. More precisely, our evaluation set contained the following documents: "La Liberté 2009/01/05" (4 pages, 172 text blocks), "La Liberté 2009/01/06" (5 pages, 275 text blocks), "La Liberté 2009/01/07" (5 pages, 302 text blocks), "La Liberté 2009/01/08" (5 pages, 263 text blocks), "La Liberté 2009/01/09" (5 pages, 255 text blocks).

As usual, we used Dolores to interactively infer a new document model by labeling the whole first newspaper, i.e., "La Liberté 2009/01/05". Figure 8.11 shows the learning evolution of this document model by reporting the recognition rates relatively to the number of labeling

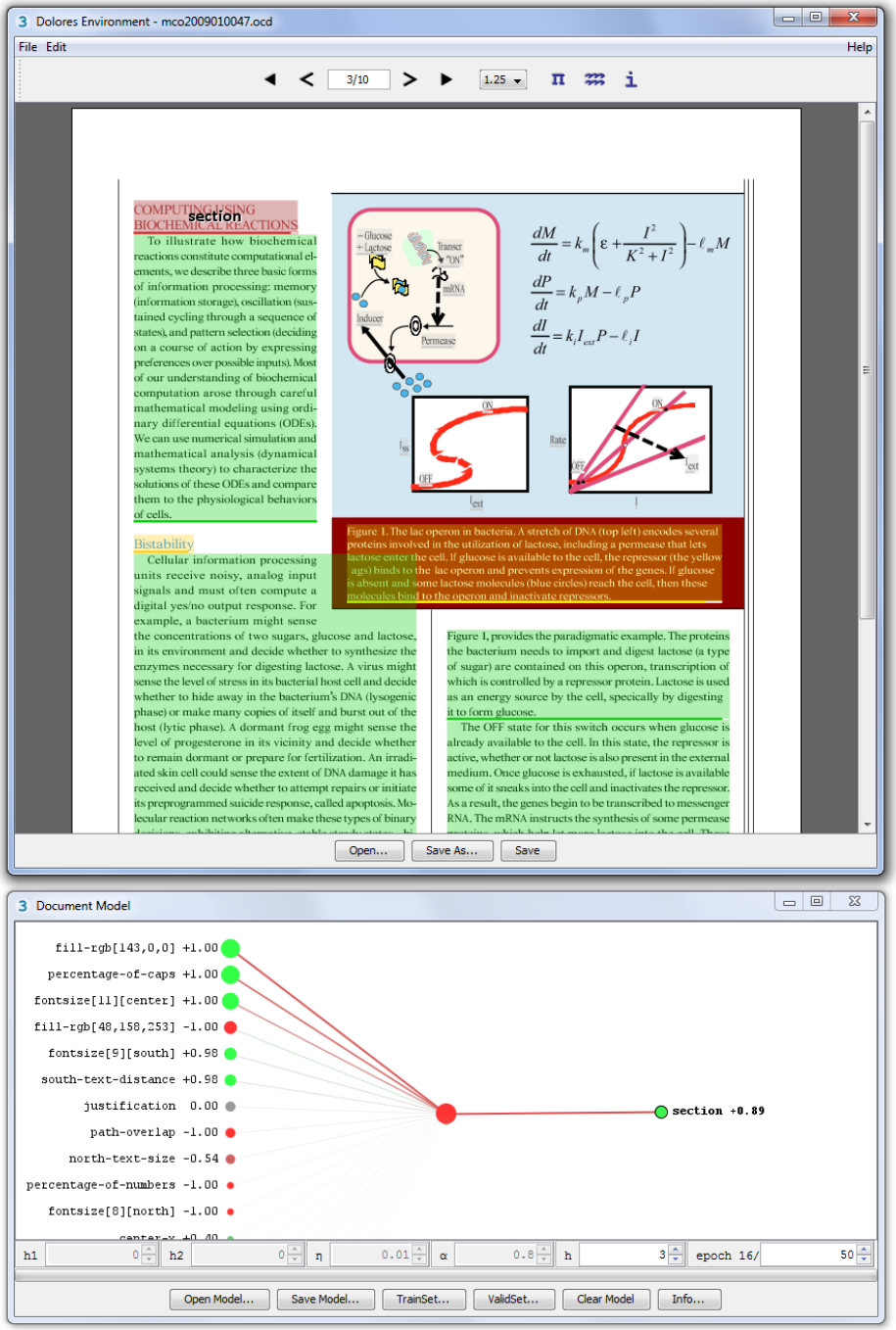


Figure 8.10: The interface of Dolores displaying a scientific paper on its labeling window and showing its most pertinent features relatively to the "section" label on its document model window.

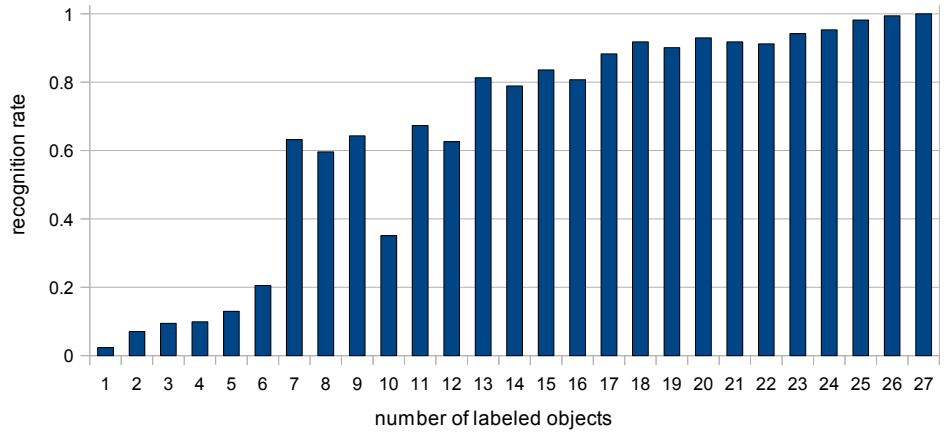


Figure 8.11: The complete integration and recognition of the first newspaper "La Liberté 2009/01/05" in the document model is achieved after the labeling of 27 objects.

actions. This first document model version was then applied on the the second newspaper "La Liberté 2009/01/06", in order to appreciate its accuracy. A recognition rate of 90.8% was achieved on it; Table 8.4 shows the corresponding classification matrix.

Part of misclassified text blocks were hence integrated in an updated version of the model, i.e., incremented with the second newspaper. This model was then applied on the third newspaper, i.e., "La Liberté 2009/01/07"; the recognition rate increased up to 96.0%. Once again, the document model was incremented with misclassified text blocks in order to obtain a better newspaper model. Finally, the resulting model was applied on the two remaining newspapers of our set, "La Liberté 2009/01/08" and "La Liberté 2009/01/09". Recognition rates of 100% and 98.8% were obtained, respectively.

We observed that the last newspaper had three misclassified text blocks: one "advertisement" text block was classified as "author" and two "caption" were classified as "page-nb". These errors are easily interpretable; indeed, advertisements have irregular layouts using lots of various font sizes and styles, which may induce the document model in error. Furthermore, in Figure 8.12, we can see that the two misclassified captions are composed of single numbers, a property that is precisely used as a discriminant feature for the "page-nb" label class (see Figure 8.13).

8.8 Document Class Recognition

Until now, we always applied document models presupposing that we knew the class of a given document. An interesting issue would be to identify the model of a document given a predefined set of document models. Each model would then be applied on a document to recognize and the model maximizing an overall confidence value would be the chosen one.

Label	Predicted x Actual Label													Total
page-nb	5	0	0	0	0	0	0	0	0	0	0	0	0	5
header	0	10	0	0	0	0	0	0	0	0	0	0	0	10
date	0	0	5	0	0	0	0	0	0	0	0	0	0	5
column	0	0	0	2	1	0	0	0	0	0	0	3	0	6
highlight	0	0	0	0	18	0	0	0	0	0	0	0	0	18
title	0	0	0	0	1	23	1	0	0	0	0	1	4	30
content	0	0	0	0	2	2	144	0	0	0	0	6	0	154
caption	0	0	0	0	0	0	0	7	0	0	0	0	0	7
subtitle	0	0	0	0	0	0	0	0	6	0	0	0	0	6
author	0	0	0	0	0	0	0	0	0	8	0	1	0	9
intercontent	0	0	0	0	0	0	0	0	0	0	1	0	0	1
advertisement	0	0	0	0	0	0	0	0	0	0	0	9	0	9
intertitle	0	0	0	0	3	0	0	0	0	0	0	0	10	13

Table 8.4: Classification matrix of "La Liberté 2009/01/06" using the model generated with "La Liberté 2009/01/05". The document model is still subject to improvements since the recognition rate is equal to 90.8%.

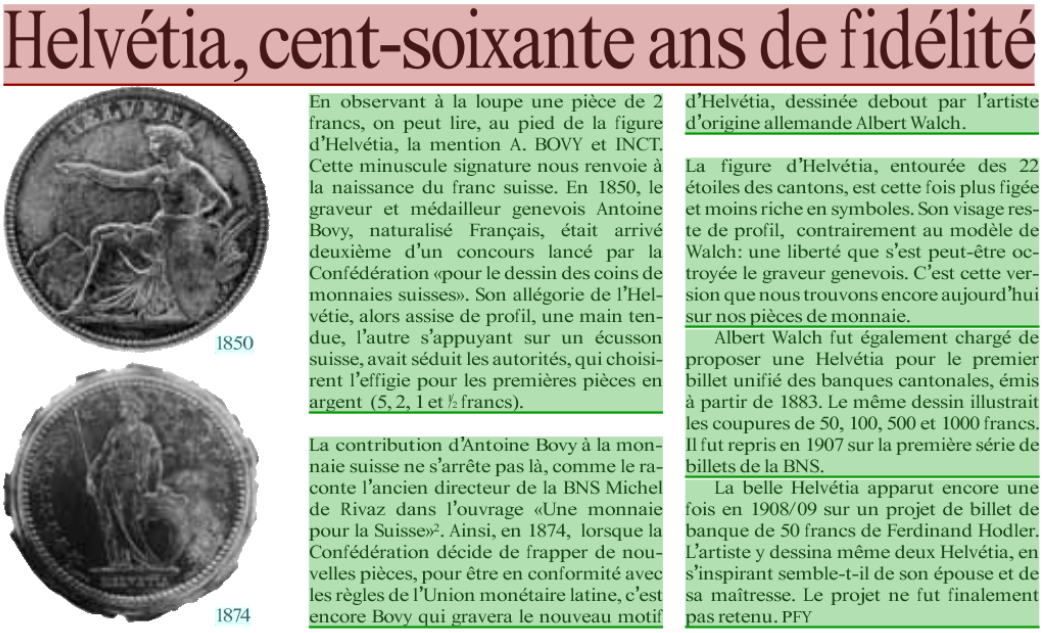


Figure 8.12: The newspaper document model misclassified two "caption" text blocks as "page-nb". The above screenshot highlights the relative complexity of the task, since the two image captions are only composed of single numbers, a property that is precisely used as a discriminant feature for "page-nb" labels.

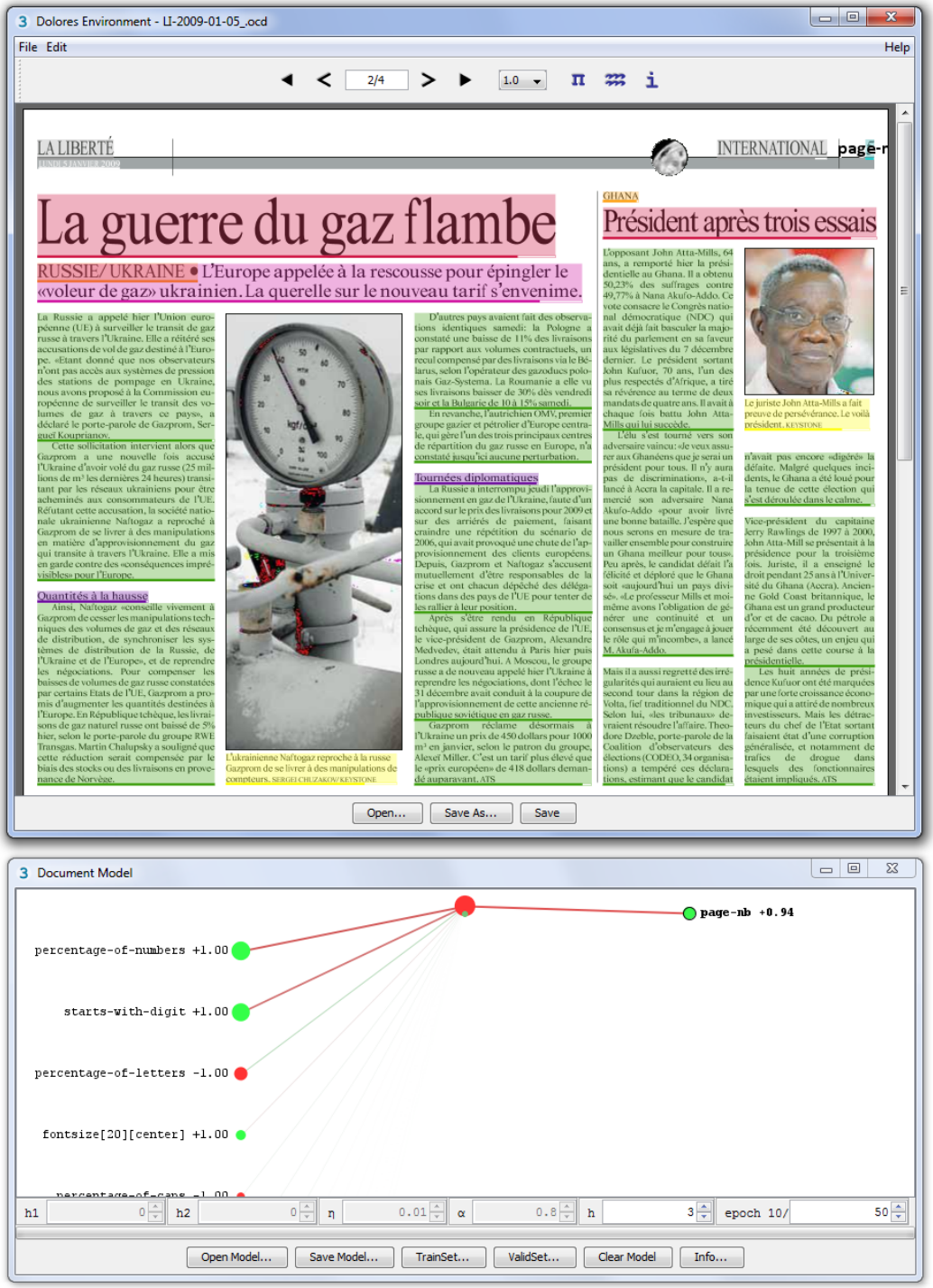


Figure 8.13: The above screenshot shows the labeling and document model windows of Dolores trained with the newspapers class. The most pertinent features relatively to the "page-nb" label concern the presence of numbers text blocks which may explain the "caption" misclassifications exposed in Figure 8.12.

Thus, we defined a new criterion called the *model confidence* which expresses the degree of confidence between a document and a potential document model. This value is always comprised between 0.0 and 1.0 and is computed as follows: first, we apply a document model on a document and retrieve the classification confidence value for each text block; the mean confidence value for each label class is then computed. As a consequence, the model confidence equals the weighted sum of the mean confidence values, knowing that each weight corresponds to the prior probability of a label class (inferred from the training set).

	TV Schedule	E-book	Wikipedia Article	Scientific Paper	Newspaper
TSI2 2005/09/21	**0.86**	0.18	0.26	0.23	0.06
TSR1 2005/09/21	**0.94**	0.15	0.30	0.23	0.06
Lawrence Sons and Lovers	0.41	**0.98**	0.11	0.28	0.07
Anna Karenina	0.41	**0.98**	0.11	0.31	0.09
Charles Breton	0.38	0.36	**0.96**	0.34	0.24
Albert Camus	0.39	0.30	**0.95**	0.32	0.23
The Organization and ...	0.41	0.53	0.52	**0.92**	0.45
Computing with Proteins	0.37	0.58	0.53	**0.93**	0.43
La Liberté 2009/01/08	0.36	0.37	0.54	0.47	**0.89**
La Liberté 2009/01/09	0.37	0.39	0.56	0.47	**0.86**

Table 8.5: A document model can be identified by computing the model confidence for each untrained document relatively to each document model.

Therefore, we computed the model confidences with the 5 previously inferred document models applied on 10 untrained documents (given our evaluation protocol, at least 2 untrained documents per class remain). The resulting values are reported in Table 8.5. We observe that the model confidence for a document is always the highest when computed with its corresponding document model. This experience is crucial because it proves that the class of a document can be identified given a set of predefined document models. Such results open the field for a fully automatic restructuring system based on a set of document models pre-inferred with Dolores.

8.9 Conclusion

This chapter has presented experimental results about the interactive and dynamic generation of document models with Dolores. These experiments first highlighted the relevance of our canonical format as a pivot format for further document logical restructuring. It also demonstrated that the canonical document representation was more than a pure pivot format and that it was an adequate representation for rapid prototyping of standalone logical structure reconstruction tools.

The various results obtained during the evaluation campaign underlined the potentiali-

ties of Dolores. For instance, only few labeling actions were sufficient to generate complete document models. Moreover, since the application of a model is always subject to errors, misclassified text blocks could be integrated at any time in the model by incrementing it.

Finally, the last experiment showed that it was even possible to identify the class of a document given a set of predefined document models. This last result establishes that Dolores is able to recognize document structures in a fully automatic way. Thereby, this thesis proposes a fully automatic document recognition system.

Lots of other interesting ideas arose during the development process of Dolores. However, due to time constraints, many of them have never been applied or even tested. Nevertheless, we still keep them under the hood as potential future works. For instance, one can imagine to create a generic newspaper model that would be well suited for a further description of each newspaper class. Then, a specific newspaper model could be inferred by adapting the existing generic newspaper model. This method would have two main advantages: first, it would produce a generic model that Dolores could apply to each class of newspapers and second, the cost in time and resources for producing a specialized model would certainly be reduced.

Concerning the input features used to infer document models, Dolores provides a mechanism to manually add new features such as regular expressions and rectangular regions. However, we could certainly improve our recognition system by dynamically generate such features. For instance, rectangular region features might be inferred from the training set by intersecting all text block bounding boxes for each label class. A similar procedure could also be applicable to regular expressions.

9

Conclusions and Perspectives

Contents

Throughout this thesis, we presented a complete and standalone system for the physical and logical structure recognition of PDF documents. This final chapter concludes our dissertation. First, Section 9.1 displays the main achievements resulting from our doctoral researches; Section 9.2 then exposes a set of potential improvements together with some future perspectives.

9.1 Achievements

A key advantage of disseminating electronic documents through PDF resides in the fact that the Adobe's file format reproduces perfectly the appearance of any source document. However, even if the Adobe's PDF specification provides means to incorporate structural information, a strong drawback of PDF files is their lack of physical and logical structures induced by poor PDF writers. Indeed, when editing or converting documents into the PDF file format, original files tend to partially or even completely lose their physical and logical structures (if existing). Thereby, a major challenge in the electronic document field is to develop techniques allowing their original structures to be recovered, a process called document reverse engineering.

In this thesis, we precisely elaborate a complete and innovative electronic document analysis system able to recover physical and logical structures from electronic documents. Our restructuring approach is generic as well as flexible, and our strategy is applicable on any composite document constituted of text, images, and graphics. The restructuring workflow is built on a two-stages process. The first phase is handled universally by XED which recovers the physical structure and generates a canonical document whose format is based on XML.

On the opposite, the second phase must be tailored to the targeted application as well as the related document class. It is handled by Dolores, a dynamic learning environment which extracts the logical structure from the physical one thanks to interactively inferred document models.

The experiments and evaluations carried out during our researches, and presented in this thesis, show that XED and Dolores bring a strong contribution to the field of document analysis. In particular, our solution distinguishes from existing document analysis systems and researches by its great usability and flexibility. Hence, the rest of this section precisely emphasizes five major contributions we have brought to our research community.

- First, the physical structure recognition handled by XED benefits from a new and original **hybrid restructuring algorithm** which is uniquely based on the electronic content of documents. Our physical restructuring methodology is an innovative approach which includes a set of original bottom-up and top-down steps using a minimal set of textual properties and dynamic thresholds.

- Second, XED stores the result of its physical restructuring process together with the document content and layout in a **canonical document format** called OCD. OCD is a complete and standalone canonical XML format that fills in the gap between document recognition formats and document presentation formats by preserving the exact appearance of original electronic documents while keeping a clear, unique, and non ambiguous description of its content, layout, and structures. Furthermore, the OCD file format optimizes the file size by precisely benefitting from its internal structured content. Our experiments have also shown that the logical restructuring task was greatly facilitated by the use of OCD. Additionally, a logical version, called OCD+, has been designed in order to store both physical and logical structures in a single representation.

- Third, a **dynamic and interactive learning system**, called Dolores, has beed developped in order to infer restructuring processes allowing logical structures to be recovered from the canonical document format. Dolores presents a document analysis architecture that allows document models to be interactively setup and refined according to user interactions. Indeed, Dolores dynamically adapts itself to the information injected by the user (learning through user validation, correction, and feedback) in order to infer a document model. Therefore, our system does not need any corpus of annotated data since they are progressively inferred and integrated in document models, thanks to the interactive and dynamic learning features of Dolores.

- Fourth, another major contribution we have exposed in this thesis is the elaboration of a **dynamic multilayer perceptron**: a learning machine which improves standard artificial neural networks by a) removing any manual configuration and b) revealing its internal functioning and knowledge through an interactive and dynamic user interface.

- Last but not least, this thesis proposes a **fully automatic document recognition system**. Indeed, Dolores is able to work in a fully self-regulating way since the class of a document can be automatically identified at runtime, thanks to a set of predefined document models.

9.2 Perspectives

As presented in this thesis, we propose a complete reverse engineering workflow operated by two complementary tools, XED and Dolores, which are used for the physical and logical structure recognition, respectively. However, the logical structure recognition achieved by Dolores is entirely based on logical labeling, which means that the logical hierarchy is not yet taken into account. As a result, the logical structure recovered with Dolores is currently represented as a flat set of labeled text blocks. Henceforth, a major issue would be to tackle the reconstruction of the logical hierarchy of electronic documents in order to take into account relationships between labels. While recovering such a hierarchical structure is straightforward for simple document layouts, i.e., having one or two columns, it is definitely far more complicated with documents having complex layouts and deep logical structures (e.g., newspapers).

Another possible improvement would be to recognize logical structures thanks to a finer granularity of textual entities. Indeed, one can distinguish two levels in the structure of a document: the microstructure and the macrostructure. The microstructure describes low level entities and their relations (characters, tokens). In contrast to this, the macrostructure can be viewed as higher level entities and their relationships such as paragraphs, list items, lists, or other textual blocks. Dolores is currently based on the macrostructure, since it uses canonical text blocks for its logical structure recognition. Nevertheless, the microstructure could as well be used as a complementary input to our recognition system, e.g., canonical tokens and/or lines. Such a technique could correct potential physical structure recognition errors and improve the logical structure granularity.

A great perspective of our restructuring workflow concerns the recognition of logical structures from paper documents. Indeed, XED and Dolores have been elaborated while keeping in mind a very generic structure recognition workflow. Since XED is able to process PDF documents containing OCRized scanned documents, it is also possible to recover their physical structure. Some previous tests precisely revealed that such documents were correctly restructured by XED, as far as the OCR system itself delivered accurate results. Dolores could then be used transparently to recognize the logical structure of OCD documents extracted from OCRized PDF files.

To conclude, it is essential to reassert that OCD is more than a pure pivot format: it is a structured, compact, and easily readable format which can be used as a standalone document presentation format. Furthermore OCD+ is intended to store both physical and

logical structures; thereby, it can also be used as a complete logical document format, thus re-enabling the lifecycle of documents by allowing them to be reedited, restyled, and reflowed. Document indexation and retrieving may greatly benefit from the easy access to OCD+ structures and content. Finally, OCD+ can be converted back to tagged PDF documents containing logical annotations.

Appendix

A

Detailed Specification of the OCD File Format

Contents

The optimized canonical document format, or OCD, is an XML-based file format used to store accurately electronic documents, i.e., content, layout, and structures. OCD has been conceived to store the physical structure of documents while keeping in mind further high-level use. Hence, thanks to the *g* markup element, it is possible to store high-level annotations, such as logical structures; logically annotated OCD files are then called OCD+.

OCD is the direct descendant file format of XCD, a first XML-based version of the canonical document format; however, practical uses showed that XCD was quite "memory wasting" due to a verbose and redundant representation scheme. As a consequence, OCD was developed to optimize the file size of XCD in order to solve its memory waste issue. At present time, we do not use XCD anymore, since it is an obsolete version of our canonical document format. As a result, we do not provide any detailed specification of XCD in the current dissertation.

This appendix is divided in two sections. First, Section A.1 presents the specification of the OCD/OCD+ file format in the form of a DTD, some explanations about the DTD itself are integrated in it as a DTDDoc. Section A.2 then exposes the XML content of a basic OCD file together with its graphical output.

A.1 Formal Definition of OCD

The formal definition of OCD is presented in details in Listing A.1 in the form of a full DTD - Document Type Definition - which defines the OCD document structure, i.e., a hierarchy

of XML markup elements together with their attributes.

Listing A.1: The full DTD declaration specifying the OCD XML-based file format.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
This DTD is commented using DTDDoc.
@title OCD Specification
@root ocd

OCD - optimized canonical document - is a complete and concise electronic
    document format used to store accurately text, images, and graphics,
    together with their layout and structures. An OCD file is an XML version
    of a canonical document, a formalism that is precisely used to express the
     exact content, layout, and structures of electronic documents in the form
     of a tree structure of canonical objects. Thus, an XML formalism is well
    suited to express a canonical document thanks to its hierarchy of XML
    markup elements. The current DTD specification describes the OCD file
    format; however, the DTD itself is not able to transcribe the use, the
    syntax, and the constraints about the OCD markup elements, their
    attributes and #PCDATA. Hence, these are explained and detailed further in
     the DTDDoc.

As specified by its name, OCD optimizes the file size by (a) using resource
    objects, (b) using compact descriptions, (c) minimizing XML attributes
    redundancy, and (d) applying a ZIP compression to the entire XML file:

(a) An OCD file contains a <b>document resources</b> section which allows
    recurrent graphical primitives to be declared only once and eventually
    referenced by means of IDs. Note that it is the role of the OCD writer to
    ensure that identical graphical primitives are declared only once in the
    document resources.

(b) The XML language is intrinsically verbose. Despite that, OCD uses <b>
    compact descriptions</b> to represent each graphical object. Images are
    stored as PNG or JPG image files and referenced by means of IDs, graphics
    are represented using an SVG-like formalism, and text is organized into a
    hierarchy of text blocks, lines, and tokens that use concise descriptions
    and benefit of a page graphics state.

(c) In OCD, minimizing the information redundancy is partly achieved through
    the use of a page <b>graphics state</b>, i.e., a simple mechanism used to
    memorize the values of the graphical element attributes while writing the
    XML output. That is, XML attributes have default values and, as long as an
     attribute value stays unchanged, it is not explicitly written in the OCD
    file. Indeed, each attribute value is memorized in the graphics state and
    is therefore implicit. An attribute is explicitly written in the OCD file
    only if its value has changed, the graphics state is then updated in
    consequence.

(d) An OCD file is compressed by using the <b>ZIP standard</b>. The XML file
    is simply "zipped" and the resulting file extension ".zip" is updated to "
    .ocd". Note that in order to ensure a full portability of the OCD file
    format, the XML encoding scheme is restricted to the ASCII - American
    Standard Code for Information Interchange - character set. Moreover, while
     it is not a restriction, OCD does not make any use of uppercase letters
    to express OCD markup element names as well as attributes and #PCDATAs.
```

Concerning the syntax of the XML attributes and #PCDATAs which are used to store the actual data of the document, both single values and arrays of values are used. Values of an array are always separated by single white space characters (the ASCII/Unicode decimal value 32). The set of characters used to encode an OCD file is therefore quite limited, thus improving the ZIP compression of the XML stream.

Before giving the DTD specification itself, we first detail some key and recurrent objects: (1) OCD colors, (2) OCD paths, and (3) OCD transform matrices.

(1) An OCD color is defined by using an array of one, three, or four real numbers separated by white spaces. The color components are of course represented in the color space used by the canonical document description, i.e., an scRGB color space together with an alpha value as last component (if existing). In case of grayscale color (saturation at 0%), only the first component is used to express the gray intensity. For instance, a pure red color with a semi-transparent alpha value would be "1 0 0 0.5" whereas a pure black would be "0". Note that the use of real values in the scRGBA colorspace enables OCD documents to work in a wide colorspace with accurate colors.

(2) An OCD path, used for vector graphics, is represented by real value operands prefixed by a set of five predefined operators: <code>m</code>, <code>l</code>, <code>c</code>, <code>q</code>, and <code>z</code>, standing for "move to", "line to", "cubic to", "quadratic to", and "close path", respectively. Actually, an OCD path is very similar to an SVG path, except that it uses only a restricted set of the SVG path operators (the ones precited). Moreover, OCD path operands are real numbers that are always represented as relative values (relatively to the previous operand), except for the first "move to" of a path. Operators and operands are separated by single white space characters.

(3) An OCD transform matrix is described thanks to four attributes <code>scale</code>, <code>shear</code>, <code>x</code>, and <code>y</code>. These attributes are used to express the scale, shear, x- and y-translation applied to graphical primitives (knowing that rotation is a combination of both scale and shear operations). Thus an OCD transform matrix represents an affine transformation using homogeneous coordinates (2D vectors are augmented with a constant value equal to 1) in order to express translations with matrix multiplication. The <code>scale</code> attribute contains two real numbers specifying the x- and y- scale operations. In case of equal x- and y- scale factors, a single value is displayed. The <code>shear</code> attribute is also constituted of two real numbers specifying the x- and y- shear operations. Again, in case of equal x- and y- shear transforms, a single value is displayed. For instance, let scaleX, scaleY, shearX, shearY, X, and Y compose an OCD transform matrix used to transform the source coordinates (x,y) into destination coordinates (tx,ty). The transform operation is achieved by simply multiplying the source coordinates (represented as a column vector) by the transform matrix itself:

```
  [tx]   [scaleX  shearX  X][x]   [scaleX*x + shearX*y + X]
  [ty] = [shearY  scaleY  Y][y] = [shearY*x + scaleY*y + Y]
  [1 ]   [  0       0     0][1]   [           1           ]
-->
```

```
<!-- The ocd element is the root element of an OCD file -->
<!ELEMENT  ocd  (resources, pages)>

<!--
@attr dpi CDATA Real value defining the resolution in DPI - dots per inch - in
     which all the position and dimension values of the documents are
    specified.
@attr date CDATA Array of three integer values in the form "yyyy mm dd" that
    represent the creation date of the document.
@attr time CDATA Array of three integer values in the form "hh mm ss" that
    represent the creation time of the document.
-->
<!ATTLIST  document
  dpi  CDATA  #REQUIRED
  date  CDATA  #REQUIRED
  time  CDATA  #REQUIRED
>

<!-- A resources element lists all the resources contained in the OCD file -->
<!ELEMENT  resources  (fonts, clips, pool?)>

<!-- A fonts element lists the fonts used in the document -->
<!ELEMENT  fonts  (font*)>

<!-- A font element is part of the resources of a document and describes a
    typeface. It represents its glyphs in a point size of 1 (knowing that
    there is 72 points per inch). -->
<!ELEMENT  font  (glyph+)>

<!--
@attr id CDATA String identification string used by a text element to
    reference its font.
@attr name CDATA String of characters specifying the name of the font.
@attr ascent CDATA Real value specifying the distance from the baseline to the
     highest ascending glyph in the typeface (the top of the glyph).
@attr descent CDATA Real value specifying the distance from the baseline to
    the lowest descending glyph in the typeface (the bottom of the glyph).
-->
<!ATTLIST font
  id  ID  #REQUIRED
  name  CDATA  #REQUIRED
  ascent  CDATA  #REQUIRED
  descent  CDATA  #REQUIRED
>

<!-- A glyph element describes a typeface glyph thanks to vector graphics. It
    contains the OCD path description of the glyph in a point size of 1-->
<!ELEMENT  glyph  (#PCDATA)>

<!--
@attr code CDATA Hexadecimal value representing the unicode character of the
    glyph.
@attr width CDATA Integer value specifying the width of the character glyph
    using a point size of 1.
@attr wind CDATA String of characters either equal to "evenodd" or "nonzero"
```

```
    which defines the winding rule used to fill the glyph path.
-->
<!ATTLIST  glyph
  code  CDATA  #REQUIRED
  width  CDATA  #IMPLIED
  wind  CDATA  #IMPLIED
>

<!-- A clips element lists the clipping paths used in the document -->
<!ELEMENT  clips  (clip*)>

<!-- A clip element is present in the resource part of an OCD document, it
    describes a clipping path used by graphics primitives to define their
    drawing region. It contains the OCD path description of the clipping path.
    -->
<!ELEMENT  clip  (#PCDATA)>

<!--
@attr id ID String identification string used by text, image, and path
    elements to reference their clipping paths.
@attr wind CDATA String of characters either equal to "evenodd" or "nonzero"
    which defines the winding rule used to determine the interior and exterior
     of the clipping path.
-->
<!ATTLIST  clip
  id  ID  #REQUIRED
  wind  CDATA  #IMPLIED
>

<!-- A pool element is present in the resource part of an OCD document and
    contains a set of recurrent graphics elements in the document -->
<!ELEMENT  pool  (g+)>

<!-- A g element groups a set of text, image, path, and g elements -->
<!ELEMENT  g  (image | path | text | g)+>

<!--
@attr id ID String identification string used to reference g elements in the
    pool of objects.
@attr g CDATA String value referencing a g element in the pool of objects.
@attr type CDATA String value defining the type of a group, e.g., "block", "
    line", "label", etc.
@attr name CDATA String value defining the name of a group and specifically
    used in OCD+ as label name.
@attr confidence CDATA Real value defining the degree of certainty relative to
     an OCD+ label.
-->
<!ATTLIST  g
  id  ID  #IMPLIED
  g  CDATA  #IMPLIED
  type  CDATA  #IMPLIED
  name  CDATA  #IMPLIED
  confidence  CDATA  #IMPLIED
>

<!-- A pages element lists the pages contained in the document -->
```

```
<!ELEMENT  pages  (page+)>

<!-- A page element is composed of text, image, path, and g elements -->
<!ELEMENT  page  (image | path | text | g)*>

<!--
@attr width CDATA Integer value specifying the width of the page (using the
    current DPI resolution).
@attr height CDATA Integer value specifying the size of the page (using the
    current DPI resolution).
-->
<!ATTLIST  page
  width  CDATA  #REQUIRED
  height  CDATA  #REQUIRED
>

<!-- An image element references a JPG or PNG image file being part of the OCD
     ZIP archive.
<!ELEMENT  image  (#PCDATA)>

<!--
@attr id CDATA String value referencing the id of an image contained in the
    resources of the document (i.e., in the ZIP archive).
@attr clip CDATA String value referencing the id of a clipping path contained
    in the resources of the document.
@attr width CDATA Integer value defining the width of the bitmap image.
@attr height CDATA Integer value defining the height of the bitmap image.
@attr scale CDATA Pair of real values defining the x- and y- scale factors; a
    single value is used in case of equal x- and y- scale factors.
@attr shear CDATA Pair of real values defining the x- and y- shear transforms;
     a single value is used in case of equal x- and y- shear transforms.
@attr x CDATA Real value defining the x-coordinate of the image.
@attr y CDATA Real value defining the y-coordinate of the image.
-->
<!ATTLIST  image
  id  ID  #REQUIRED
  clip  CDATA  #IMPLIED
  width  CDATA  #IMPLIED
  height  CDATA  #IMPLIED
  scale  CDATA  #IMPLIED
  shear  CDATA  #IMPLIED
  x  CDATA  #IMPLIED
  y  CDATA  #IMPLIED
>

<!-- A path element describes vector graphics, it contains the OCD path
    description -->
<!ELEMENT  path  (#PCDATA)>

<!--
@attr id CDATA String value referencing the id of a path contained in the
    resources of the document.
@attr clip CDATA String value referencing the id of a clipping path contained
    in the resources of the document.
@attr scale CDATA Pair of real values defining the x- and y- scale factors; a
    single value is used in case of equal x- and y- scale factors.
```

```
@attr shear CDATA Pair of real values defining the x- and y- shear transforms;
     a single value is used in case of equal x- and y- shear transforms.
@attr x CDATA Real value defining the x-coordinate of the path.
@attr y CDATA Real value defining the y-coordinate of the path.
@attr fill CDATA Array of four real values representing an OCD color used to
    fill the interior of the path.
@attr stroke CDATA Array of four real values representing an OCD color used to
     draw the outline of the path.
@attr pen CDATA Real value defining the size of the "pen", i.e., the outline
    of the path.
@attr cap CDATA String of characters defining the decoration applied to the
    ends of unclosed subpaths and dash segments, it can have the following
    values: "butt", "round", and "square".
@attr join CDATA String of characters defining the decoration applied at the
    intersection of two path segments, it can have the following values: "
    bevel", "miter", and "round".
@attr wind CDATA String of characters either equal to "evenodd" or "nonzero"
    which defines the winding rule used to fill the path.
@attr dash CDATA Array of real values specifying the dash pattern used to draw
     the outline of the path.
@attr phase CDATA Real value specifying the dash phase used to draw the dash
    pattern of the path.
-->
<!ATTLIST  path
  id  ID  #IMPLIED
  clip   CDATA  #IMPLIED
  scale  CDATA  #IMPLIED
  shear  CDATA  #IMPLIED
  x  CDATA  #IMPLIED
  y  CDATA  #IMPLIED
  fill  CDATA  #IMPLIED
  stroke  CDATA  #IMPLIED
  pen  CDATA  #IMPLIED
  cap  CDATA  #IMPLIED
  join  CDATA  #IMPLIED
  wind  CDATA  #IMPLIED
  dash  CDATA  #IMPLIED
  phase  CDATA  #IMPLIED
>

<!-- A text element describes an OCD text token , it contains an array of
    hexadecimal values representing the character unicodes of the token -->
<!ELEMENT  text  (#PCDATA)>

<!--
@attr clip CDATA String value referencing the id of a clipping path contained
    in the resources of the document.
@attr x CDATA Real value defining the x-coordinate of the text.
@attr y CDATA Real value defining the y-coordinate of the text.
@attr scale CDATA Pair of real values defining the x- and y- scale factors; a
    single value is used in case of equal x- and y- scale factors.
@attr shear CDATA Pair of real values defining the x- and y- shear transforms;
     a single value is used in case of equal x- and y- shear transforms.
@attr fill CDATA Array of four real values representing an OCD color used to
    fill the interior of the font glyphs.
@attr font CDATA String value referencing the id of a font contained in the
```

```
    resources of the document.
@attr cs CDATA Array of real values separated by white spaces and defining the
     character spacings, i.e., the delta values adjusting the default
    character widths; the array is trimmed by removing equal values from its
    end.
@attr tx CDATA Real value defining the token spacing, i.e., the delta value
    adjusting the current white space width.
@attr ty CDATA Real value defining the text rising, i.e., the delta value
    adjusting the current baseline y-coordinate.
@attr lx CDATA Real value defining the current line indentation, i.e., the
    distance between the current line x-coordinate and the previous one.
@attr ly CDATA Real value defining the current line spacing, i.e., the
    distance between the current baseline and the previous one.

@attr ws CDATA Real value defining the width of the current white space
    character.
-->
<!ATTLIST  text
  clip   CDATA  #IMPLIED
  x  CDATA  #IMPLIED
  y  CDATA  #IMPLIED
  scale  CDATA  #IMPLIED
  shear  CDATA  #IMPLIED
  fill  CDATA  #IMPLIED
  font  CDATA  #IMPLIED
  cs  CDATA  #IMPLIED
  tx  CDATA  #IMPLIED
  ty  CDATA  #IMPLIED
  lx  CDATA  #IMPLIED
  ly  CDATA  #IMPLILED
  ws  CDATA  #IMPLIED
>
```

A.2 A Concrete OCD Example

A tiny OCD example is presented in Listing A.2 in the form of its decompressed XML source. Figure A.1 displays the graphical output of the tiny OCD example.

Listing A.2: The decompressed XML source corresponding to the OCD example displayed in Figure A.1.

```
<?xml version="1.0" encoding="utf-8"?>
<ocd dpi="72" date="2010 06 05" time="09 56 45">
 <resources>
  <fonts>
   <font name="Helvetica" ascent=".718" descent=".207">
    <glyph code="20" width=".278">m 0 0</glyph>
    <glyph code="44" width=".722">m .081 0 l 0 -.718 l 0 0 l .048 0 l 0 0 l
        .247 0 q .137 0 .218 .097 q .081 .097 .081 .262 q 0 .164 -.083 .261 q
        -.083 .097 -.223 .097 l -.239 0 l 0 0 l -.048 0 l 0 0 z m .097 -.083 l
         0 0 l .178 0 q .104 0 .161 -.072 q .057 -.072 .057 -.204 q 0 -.132
        -.055 -.204 q -.055 -.072 -.155 -.072 l -.186 0 l 0 0 l 0 .552 z</
        glyph>
```

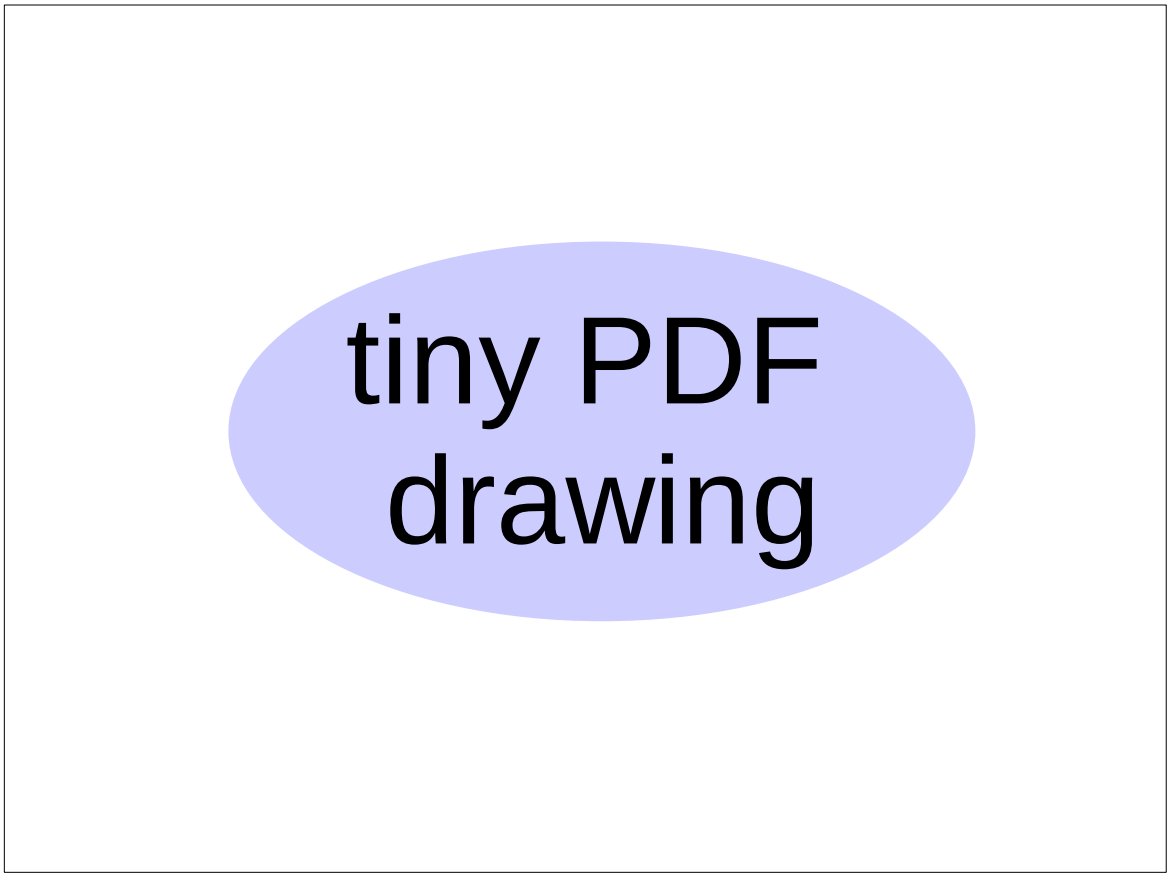


Figure A.1: This figure displays the graphical output of the OCD file presented in Listing A.2

```
<glyph code="46" width=".611">m .086 0 l 0 -.718 l 0 0 l .497 0 l 0 0 l 0
    .043 l 0 .043 l 0 0 l -.4 0 l 0 0 l 0 .219 l 0 0 l .35 0 l 0 .043 l 0
    .043 l 0 0 l -.35 0 l 0 0 l 0 .327 l -.048 0 l -.049 0 z</glyph>
<glyph code="50" width=".667">m .086 0 l 0 -.718 l 0 0 l .048 0 l 0 0 l
    .266 0 q .104 0 .163 .052 q .059 .052 .059 .145 q 0 .102 -.059 .16 q
    -.059 .058 -.163 .058 q -.137 0 -.217 .001 l 0 0 l 0 .302 l 0 0 l
    -.048 0 l -.049 0 l 0 0 z m .097 -.386 l 0 0 l .193 0 q .076 0 .111
    -.032 q .035 -.032 .035 -.1 q 0 -.061 -.035 -.089 q -.035 -.028 -.111
    -.028 l -.193 0 l 0 0 l 0 .249 z</glyph>
<glyph code="61" width=".556">m .203 .014 q -.077 0 -.122 -.041 q -.045
    -.041 -.045 -.111 q 0 -.07 .042 -.112 q .042 -.042 .123 -.052 l .12
    -.015 q .015 -.001 .028 -.005 q .017 -.004 .026 -.018 q .009 -.013
    .009 -.033 l 0 -.004 q 0 -.042 -.032 -.066 q -.032 -.024 -.088 -.024 q
     -.054 0 -.085 .025 q -.031 .025 -.037 .075 l 0 0 l -.081 0 l 0 0 q
    .006 -.086 .061 -.129 q .054 -.043 .155 -.043 q .095 0 .145 .041 q .05
     .041 .05 .12 l 0 .287 q 0 .013 .008 .021 q .008 .008 .022 .008 q .004
     0 .011 -.001 q .008 -.001 .016 -.003 l 0 .063 q -.013 .005 -.027 .008
     q -.014 .003 -.024 .003 q -.05 0 -.072 -.027 q -.015 -.018 -.021
    -.051 l 0 0 q -.01 .012 -.022 .023 q -.066 .059 -.161 .059 z m .182
    -.221 l 0 -.057 l 0 0 q -.024 .012 -.054 .016 l -.081 .012 q -.063
    .009 -.093 .03 q -.029 .021 -.029 .056 q 0 .044 .024 .068 q .024 .024
    .068 .024 q .069 0 .117 -.038 q .033 -.026 .043 -.06 q .002 -.006 .003
     -.02 q .001 -.014 .001 -.031 z</glyph>
<glyph code="64" width=".556">m .264 .014 q -.105 0 -.167 -.078 q -.062
    -.078 -.062 -.212 q 0 -.118 .06 -.19 q .06 -.072 .157 -.072 q .096 0
    .152 .082 l .007 .01 l 0 0 l 0 -.271 l 0 0 l .044 0 l .044 0 l 0 0 l 0
     .718 l 0 0 l -.041 0 l -.041 0 l 0 0 l 0 -.071 l 0 0 l -.011 .016 q
    -.054 .069 -.142 .069 z m .005 -.473 q -.074 0 -.108 .047 q -.034 .047
     -.034 .15 q 0 .097 .037 .149 q .037 .052 .105 .052 q .064 0 .103
    -.048 q .039 -.048 .039 -.127 q 0 -.107 -.037 -.165 q -.037 -.058
    -.105 -.058 z</glyph>
<glyph code="67" width=".556">m .256 .148 q .076 0 .115 -.048 q .04 -.048
    .04 -.139 l 0 -.017 l 0 .001 q -.054 .069 -.142 .069 q -.105 0 -.167
    -.078 q -.062 -.078 -.062 -.212 q 0 -.118 .061 -.19 q .061 -.072 .16
    -.072 q .094 0 .149 .082 l .006 .009 l 0 0 l 0 -.075 l 0 0 l .083 0 l
    0 .484 q 0 .132 -.06 .196 q -.059 .063 -.182 .063 q -.089 0 -.141
    -.041 q -.052 -.041 -.063 -.119 l 0 0 l .09 0 l 0 0 q .008 .042 .037
    .065 q .03 .023 .075 .023 z m .017 -.606 q -.074 0 -.108 .047 q -.034
    .047 -.034 .149 q 0 .096 .037 .148 q .037 .052 .105 .052 q .064 0 .103
     -.048 q .039 -.048 .039 -.126 q 0 -.111 -.035 -.166 q -.035 -.055
    -.107 -.055 z</glyph>
<glyph code="69" width=".222">m .068 0 l 0 -.522 l 0 0 l .044 0 l .044 0 l
     0 0 l 0 .522 l 0 0 l -.044 0 l -.044 0 l 0 0 z m -.001 -.718 l .09 0
    l 0 .1 l -.09 0 l 0 -.1 z</glyph>
<glyph code="6e" width=".556">m .065 0 l 0 -.522 l 0 0 l .042 0 l .042 0 l
     0 0 l 0 .077 l 0 0 l 0 0 l .02 -.026 q .057 -.067 .143 -.067 q .086 0
     .133 .047 q .047 .047 .047 .133 l 0 .358 l 0 0 l -.044 0 l -.044 0 l
    0 0 l 0 -.328 q 0 -.069 -.027 -.099 q -.027 -.031 -.086 -.031 q -.063
    0 -.101 .048 q -.037 .048 -.037 .13 l 0 0 l 0 .28 l 0 0 l -.044 0 l
    -.044 0 l 0 0 z</glyph>
<glyph code="72" width=".333">m .077 0 l 0 -.522 l 0 0 l .041 0 l .042 0 l
     0 0 l 0 .089 l 0 0 l 0 0 q .011 -.021 .024 -.038 q .05 -.067 .123
    -.067 q .012 0 .025 .004 l 0 .09 l -.018 0 q -.071 0 -.11 .043 q -.039
     .043 -.039 .122 l 0 0 l 0 .28 l 0 0 l -.044 0 l -.044 0 l 0 0 z</
    glyph>
<glyph code="74" width=".278">m .257 0 q -.045 .007 -.065 .007 q -.058 0
```

```
        -.081 -.025 q -.024 -.025 -.024 -.087 l 0 -.344 l 0 0 l -.072 0 l 0
        -.072 l .072 0 l 0 0 l 0 -.146 l .088 0 l 0 .146 l 0 0 l .083 0 l 0
        .072 l -.083 0 l 0 0 l 0 .344 q 0 .018 .014 .028 q .014 .01 .037 .01 l
         .031 0 l 0 .068 z</glyph>
    <glyph code="77" width=".722">m .163 0 l -.148 -.522 l 0 0 l .048 0 l .048
         0 l 0 0 l .1 .416 l 0 0 l 0 0 l .102 -.416 l .049 0 l .049 0 l .105
        .419 l 0 0 l 0 0 l .106 -.419 l 0 0 l .043 0 l .044 0 l 0 0 l -.148
        .522 l -.047 0 l -.047 0 l -.108 -.408 l 0 0 l 0 0 l -.102 .408 l
        -.047 0 l -.047 0 z</glyph>
    <glyph code="79" width=".5">m .489 -.522 l -.199 .54 q -.046 .121 -.082
        .158 q -.036 .037 -.108 .037 q -.023 0 -.042 -.005 l 0 -.082 q .02
        .007 .05 .008 q .03 0 .05 -.024 q .02 -.024 .045 -.091 l .001 -.008 l
        0 0 l -.194 -.531 l 0 0 l .05 0 l .05 0 l 0 0 l .14 .432 l 0 0 l 0 0 l
         .142 -.432 l 0 0 l .048 0 l .048 0 l 0 0 z</glyph>
   </font>
  </fonts>
  <clips>
   <clip name="1">m 0 85 l 0 -85 l 113 0 l 0 85 z</clip>
  </clips>
  <pool/>
 </resources>
 <pages>
  <page width="113" height="85">
   <path x="0" y="85" clip="1" fill=".8 .8 1" pen=".1" wind="evenodd">m 58.3
       -24.9 c -19.9 0 -36 -8.2 -36 -18.3 c 0 -10.1 16.1 -18.3 36 -18.3 c 19.9
        0 36 8.2 36 18.3 c 0 10.1 -16.1 18.3 -36 18.3 z</path>
   <g type="block">
    <g type="line">
     <text x="33.6" y="39" scale="12" font="Helvetica" fill="0" cs="4 -5 1 0">
         74 69 6e 79</text>
     <text ws="282"/>
     <text cs="0 -5 0">50 44 46</text>
    </g>
    <g type="line">
     <text lx="3.6" ly="14" cs="10 -8 10 -5 -5 10 0">64 72 61 77 69 6e 67</
         text>
    </g>
   </g>
  </page>
 </pages>
</ocd>
```

B

Feedforward Artificial Neural Networks - Multilayer Perceptrons

Contents

This appendix is a complement to Chapter 7, "Dolores, Interactive Logical Restructuring", it provides a detailed introduction to artificial neural networks.

An artificial neural network - ANN -, is a network of simple virtual processing elements called neurons. Such network can implement complex global behaviour, determined by the connections between the neurons and their parameters. ANNs try to simulate the structure and functional aspects of biological neural networks, and thus they are directly inspired from the central nervous system. A multilayer perceptron - MLP -, is a specific type of artificial neural network used to describe any general feedworward network, i.e., without recurrent connections. An MLP is a really powerful type of ANN since it is a universal function approximator, as proven by the Cybenko theorem [27].

This appendix presents the theoretical background of MLPs, it is entirely based on the Duda-Hart "Pattern Classification" book [31], thus, all the definitions and formulas are extracted from this reference book. First, Section B.1 explains the fundaments of MLPs, their construction and internal mechanism (feedforward operation and classification). Then, Section B.2 exposes a standard learning technique of MLPs, i.e., the backpropagation algorithm. And finally, Section B.3 details the various parameters of an MLP and how they can be set.

B.1 Feedforward Operation and Classification

The purpose of an MLP is to classify an input vector (or sample) into a predefined label class. Thus, an MLP is a discriminant function, $z_k = g_k(\mathbf{x})$, taking as input a vector $\mathbf{x}$ in a d-dimensional space and outputting a vector $\mathbf{z}$ in a c-dimensional space. The discriminant function described by an MLP can implement arbitrary decision boundaries in a d-dimensional space, the decision regions need neither to be convex, nor to be connected.

In order to perform its task, an MLP is composed of computing units called neurons (or simply units). The output of a neuron is computed thanks to the three following elements: a set of input values, a set of links connecting the inputs to the neuron, and an internal activation function (see Figure B.1). The output of a neuron is then a function of the weighted sum of its inputs.

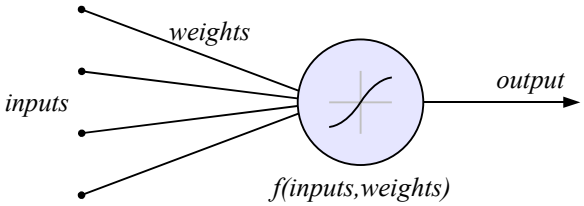


Figure B.1: A neuron is a simple computing unit. The output of a neuron is a function of the weighted sum of its inputs.

An MLP is a network of neurons, more precisely, it is composed of successive layers of neurons. The first layer of an MLP is called the input layer, the last layer is called the output layer, and the other layers are called the hidden layers (see Figure B.2). An MLP is a specific type of ANN where the information moves in only one direction, forward, from the input nodes, through the hidden nodes and to the output nodes. There are no cycles or loops in an MLP, since it is a feedforward neural network. Therefore, the feedforward operation consists of presenting a pattern to the input units and passing the signals through the network (and thus layers) in order to yield output from the output units.

During the feedforward operation, each neuron transforms a set of input values to a single output value. First, a net activation is computed, resulting from the weighted sum of its inputs. Then, the net activation is transformed in an output activation thanks to an activation function.

That is, lets d be the number of input neurons, and net_j the net activation of the j^{th} neuron of the hidden layer, then the net activation of a neuron is the inner product between its input values and input link weights:

$$net_j = \sum_{i=0}^{d} x_i w_{ji} \tag{B.1}$$

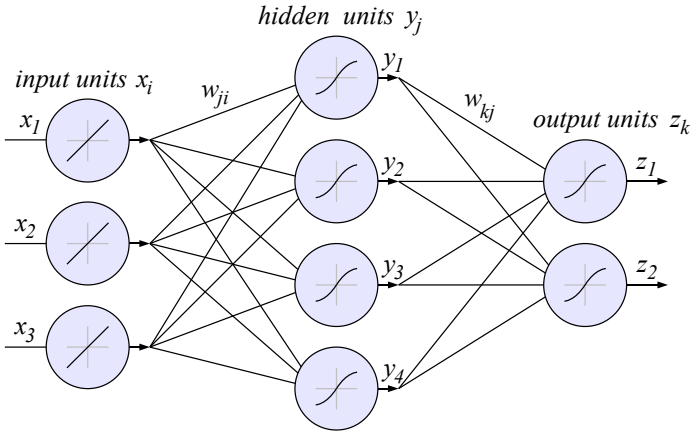


Figure B.2: A multilayer perceptron with three layers, three input units (neurons), four hidden units, and two output units.

The subscript i indexes units in the input layer, whereas the subscript j indexes units in the hidden layer. The value w_{ji} denotes the input-to-hidden layer link weight between the input unit i and the hidden unit j. In analogy with neurobioloy, such weights connections are sometimes called "synapses" and the values of the connections the "synaptic weights". The output activation of the j^{th} neuron of the hidden layer is then a function of its net activation:

$$y_j = f(net_j) \tag{B.2}$$

The Equation B.2 is generalizable to each neuron of an MLP, at any layer. Thus, suppose we have a three layered MLP with d input units, h hidden units, and c output units (corresponding to c different classes or categories). Therefore, the signal from each ouput is the result of the feedforward operation described by the discriminant function $g_k(\mathbf{x})$:

$$g_k(\mathbf{x}) \equiv z_k = f(\sum_{j=1}^{h} w_{kj} f(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}) + w_{k0}) \tag{B.3}$$

In Equation B.3 we observe that two bias units represented by the weights w_{j0} and w_{k0} have been added. A bias unit is a single neuron whose output values is a constant equal to 1.0, it is ususally added to every layer of an MLP except the output one. This bias comes from the fact that a single perceptron needs an additional constant value to be able to generate a separating hyperplane that does not pass through the origin of the space.

B.2 The Backpropagation Algorithm

The feedforward operation exposed above supposes that the link weights of the network have been previously set. The backpropagation algorithm is precisely a technique used to

infer the weights of an MLP thanks to a set of known training samples (or patterns). The backpropagation algorithm is part of the supervised learning techniques where a set of known training samples (the ground-truthed data) is used to set the parameters of a system in order to bring the actual outputs closer to the desired ones. Actually, the backpropagation is one of the simplest and most general methods for supervised learning of multilayer neural networks.

The backpropagation algorithm uses the error between the actual output and the desired output of each unit to update the network weights. The computation of the error is therefore trivial at the output layer, while this is not the case for the hidden layers. The strength of the backpropagation algorithm is precisely that it provides a straightforward way to calculate an effective error for each hidden unit, from which a learning rule may be derived for the input-to-hidden weights.

In more pratical terms, the backpropagation algorithm proceeds the following. First, a training pattern is presented to the input layer, the signals is propagated through the network thanks to the feedforward mechanism and the output activations at the ouptut layer are determined. The output values are then compared to the target values and their differences (if any) correspond to the output errors. A general output error of the network $e(\mathbf{w}, \mathbf{x})$ (see Equation B.4) is computed, it is a scalar function of the weights which is minimized when the network outputs match the desired outputs. The weights of the network are thus adjusted in order to reduce this error measure.

Lets $\mathbf{x}$ be a training input, $\mathbf{w}$ the network weights, c the number of categories (output units), $\mathbf{t}$ the desired output, and $\mathbf{z}$ the actual output, the training error on a pattern is the following:

$$e(\mathbf{w}, \mathbf{x}) \equiv \frac{1}{2} \sum_{k=1}^{c} (t_k - z_k)^2 \tag{B.4}$$

The backpropagation learning rule is based on gradient descent, therefore, the weights are changed in a direction that will reduce the error:

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}} \tag{B.5}$$

A learning rate η is used to control the relative size of the changes in weights. The rules leading to the weights update result from the backpropagation of the errors. Thus, the learning rules for the hidden-to-output weights and for the input-to-hidden weights are, respectively:

$$\Delta w_{kj} = \eta (t_k - z_k) f'(net_k) y_j \tag{B.6}$$

$$\Delta w_{ji} = \eta [\sum_{k=1}^{c} w_{kj} (t_k - z_k) f'(net_k)] f'(net_j) x_i \tag{B.7}$$

The backpropagation rules presented above correspond to a three-layer MLP, however a generalization for MLPs having any number of hidden layers is trivial.

Finally, the whole training process of an MLP is achieved by iteratively applying the backpropagation algorithm on a set of training samples. Each backpropagation iteration is called a learning *cycle* or *epoch*. The training process begins with an untrained network (the weights are initialized with random values) and iterates over the training samples until the general output error is deemed significantly low.

B.3 Defining MLP Parameters

A multilayer perceptron has various parameters that can be set: the number of hidden layers, the number of units in each hidden layer, the initialization of the weights, the learning rate, the learning momentum, the weight decay, the activation functions, the input data normalization, and the learning protocol.

B.3.1 Number of Hidden Layers

Actually, three layers are sufficient to implement any arbitrary function. Therefore, the use of more layers would need special problem conditions or requirements. For instance, it is easier for a four-layer MLP to learn translations than for a three-layer one. This is because each layer can generally easily learn an invariance within a limited range of parameters. Some functions can also be implemented more efficiently (i.e., with fewer total units) in networks with more than one hidden layer.

However, it has been found empirically that networks with multiple hidden layers are more prone to getting caught in undesireable local minima. Moreover, increasing the number of hidden layers may facilitate complex decision boundaries leading to over-fitting. In the absence of problem-specific reason, MLPs with one hidden layer are usually preferred.

B.3.2 Number of Hidden Units

While the number of input units and output units are dictated by the dimensionality of the input vectors and the number of categories, respectively, the number of hidden units is not simply related to such obvious properties of the classification problem.

The hidden units of an MLP are related to the complexity of the input densities. That is, there should be enough hidden units to express the complexity of the decision boundary. However an excessive number of hidden units would lead to an over-fitting of the training samples, thus preventing the generalization ability of the network. As a matter of fact, there is simply no general rule to set the number of hidden units.

B.3.3 Weights Initialization

Suppose that all the weights are first initialized to zero, then the backpropagated error would also be zero and the input-to-hidden weights would never change. The weights initialization

of an MLP is therefore a mandatory step. Moreover, the choice of the initial weights has a clear impact on the learning efficiency. Given normalized learning data, the following rules ensure fast and uniform learning:

$$-1/\sqrt{d} < w_{ji} + 1/\sqrt{d} \tag{B.8}$$

$$-1/\sqrt{h} < w_{kj} + 1/\sqrt{h} \tag{B.9}$$

B.3.4 Learning Rate

The learning rate η is a value determining the speed at which the network reaches a mininum in the error function. However, in order to ensure convergence, and then attain a minimum, the learning rate should be small enough. According to the Duda Hart "Pattern Classification" book [31], for rapid and uniform learning we should calculate the second derivative of the error function with respect to each weight and set the optimal learning rate separately for each weight. A learning rate of $\eta \simeq 0.1$ is often adequate as a first choice. The learning rate should then be lowered if the error function diverges during learning, or instead should be raised if learning seems unduly slow.

B.3.5 Momentum Rate

The momentum rate corresponds to a weight update inertia. Indeed, when learning, the error surfaces often have plateaus, i.e., regions in which the slope is very small. Thus momentum allows the network to learn more quickly when plateaus in the error surface exist. The approach is simply to modify the learning rule by adding some fraction α of the previous weight update (values typically used are $\alpha \simeq 0.9$):

$$\mathbf{w}(m+1) = \mathbf{w}(m) + (1-\alpha)\Delta\mathbf{w}(m) + \alpha\Delta\mathbf{w}(m-1) \tag{B.10}$$

B.3.6 Weight Decay

One method of simplifying the network and avoiding overfitting is to impose a heuristic that the weights should be small. The basic aproach is to start with a network with "too many" weights and "decay" all weights during training. Small weights favor models that are more nearly linear. Equation B.11 shows that after each weight update, every weight is simply "decayed" ($0 < \epsilon < 1$).

$$w^{new} = w^{old}(1-\epsilon) \tag{B.11}$$

B.3.7 Activation Function

The backpropagation algorithm works with any activation function, given that a few simple conditions such as continuity and derivative are met. However, the activation function

should be non-linear in order to give three-layered networks their computational power. The activation function should also saturate at some minimum and maxiumum value in order to keep the weights and activations bounded, and thus improve the training efficiency and time. Saturation is also a particularly desirable property when the output of a network is meant to represent a probability. Monotonicity and linearity for small values of *net* are also desirable properties. One class of function that has all the above properties is the sigmoid, such as the hyperbolic tangent:

$$f(net) = a\tanh(b\,net) = a\frac{e^{+b\,net} - e^{-b\,net}}{e^{+b\,net} + e^{-b\,net}} \tag{B.12}$$

By choosing $a = 1.716$ and $b = 2/3$, we ensure that $f'(0) \simeq 0.5$, that the linear range is $-1.0 < net < +1.0$, and that extrema of the second derivative occur roughly at $net \simeq \pm 2.0$ (see Figure B.3).

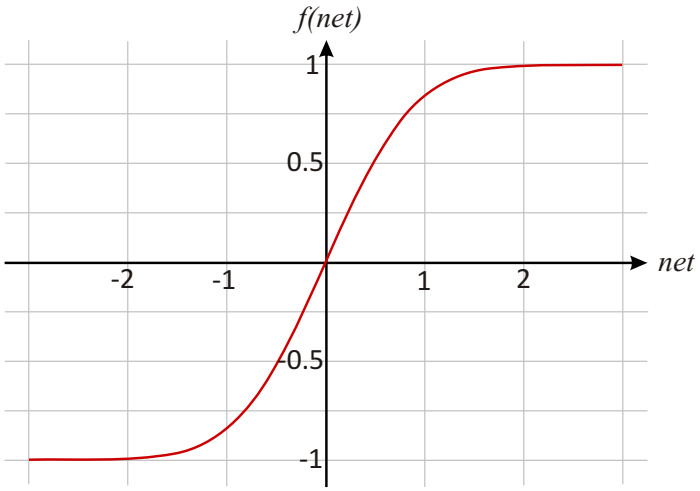


Figure B.3: A sigmoid function of the form $a\tanh(b\,net)$, where $a = 1.716$ and $b = 2/3$.

B.3.8 Training Protocols

The batchpropagation technique is commonly used with three different training protocols: stochastic, batch, and on-line. In stochastic training, patterns are chosen randomly and iteratively from the training set, the network weights are updated for each pattern presentation. In batch training, all the training patterns are presented to the network before the weights update iteration takes place. In stochastic and batch training, the process iterates over the training data until the error is deemed significantly low. At opposite, in online training, each pattern is presented only once to the network, this technique is especially used when memory becomes a problem.

B.3.9 Number of Training Cycles

Right after the weights initialization (randomization) the error on the training set is typically high. Through the learning the error becomes lower on the training set. The learning process of an MLP consists in an iterative number of learning epochs, or cycles, leading to an asymptotic error value on the training set (given that the learning rate is small enough).

However, the average error on an independent test set is virtually always higher than on the training set, and may start to increase after a certain amount of training cycles (see Figure B.4). Such effects are the result of an over-training of the network, leading to decision boundaries that overfit the training data. The MLP is then specialized to its training set and therefore lacks of generalization.

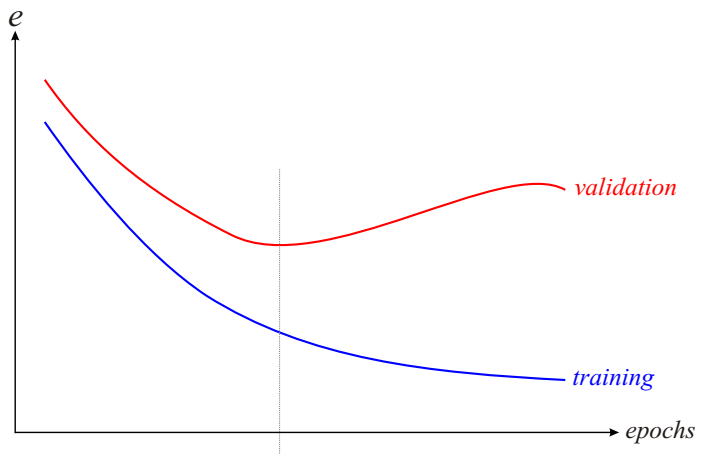


Figure B.4: During the training process, the error on a validation set may suddenly start to increase after a certain amount of training cycles.

Thus, the number of training cycles should minimize the error on a set of untrained patterns, i.e., the validation set. Moreover, the global MLP error should not be computed on a set used during the training phase, i.e, the training or validation set, but instead on a still unused set, the test set. A commonly used method for minimizing the error over untrained patterns is to split the set of training samples into various train and validation sets, this technique is called the cross-validation.

B.3.10 Data Normalization

The backpropagation algorithm performs at best when each input feature is treated equally. That is, the network should not be influenced by some input features because their representations differ in magnitude. Moreover, small and balanced input values (positive and negative) tend to avoid an excessive increase of the network weights.

Therefore, the input samples should be shifted and scaled in order to ensure that the input values are more-or-less comprised in the range [-1.0,1.0]. A solution is simply to look for the minimum and maximum value of each feature of the training set and then scale and shift them to fit in this range. Another common solution is to compute the mean (μ) and standard deviation (σ) of each feature in the training set and then normalize them, i.e., $x' = (x - \mu)/\sigma$.

Bibliography

[1] Able2Extract, 2009. http://www.investintech.com.

[2] CORPORATE Adobe Systems Inc. *PostScript language reference (3rd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] M. Aiello, C. Monz, L. Todoran, and M. Worring. Document understanding for a broad class of documents. In *IJDAR International Journal on Document Analysis and Recognition*, pages 1–16, 2002.

[4] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[5] O. Altamura, F. Esposito, and D. Malerba. Transforming paper documents into xml format with wisdom++. *International Journal of Document Analysis and Recognition (IJDAR)*, 3(2):175–198, 2001.

[6] A. Anjewierden and S. Kabel. Automatic indexing of documents with ontologies. In *13th Belgian/Dutch Conference on Artificial Intelligence (BNAIC 2001)*, pages 23–30, Amsterdam, Holland, 2001.

[7] A. Antonacopoulos. Page segmentation using the description of the background. *Computer Vision and Image Understanding*, 70(3):350–369, 1998.

[8] A. Azokly. *Une approche générique pour la reconnaissance de la structure physique de documents composites*. PhD thesis, PhD Thesis, IIUF-Université de Fribourg, 1995.

[9] S. R. Bagley, D. F. Brailsford, and M. R. B. Hardy. Creating reusable well-structured pdf as a sequence of component object graphic (cog) elements. In *ACM Symposium on Document Engineering (DocEng'03)*, pages 58–67, Grenoble, France, 2003.

[10] H.S. Baird, S.E. Jones, and S.J. Fortune. Image segmentation by shape-directed covers. In *ICDAR International Conference on Pattern Recognition*, volume 1, pages 820–825, Atlantic City, USA, 1990.

[11] F. Bapst. *Reconnaissance de documents assistée : architecture logicielle et intégration de savoir-faire*. PhD thesis, PhD Thesis, IIUF-Université de Fribourg, Switzerland, 1998. thesis Nr. 1228.

[12] A. Belaïd. Conception automatisée de modèles de page en vue de leur utilisation en reconnaissance de documents. In *Workshop on Electronic Page Models (LAMPE'97)*, Lausanne, Suisse, 1997.

[13] A. Belaïd, Y. Rangoni, and I. Falk. Xml data representation in document image analysis. In *ICDAR'07: Proceedings of the Ninth International Conference on Document Analysis and Recognition*, pages 78–82, Washington, DC, USA, 2007. IEEE Computer Society.

[14] J.-L. Bloechle, M. Rigamonti, K. Hadjar, D. Lalanne, and R. Ingold. Xcdf: A canonical and structured document format. In *7th International Workshop, DAS'06*, number 3872 in LNCS, pages 141–152, Nelson, New Zealand, February 2006. Springer-Verlag.

[15] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[16] Thomas M. Breuel. Two geometric algorithms for layout analysis. In *DAS '02: Proceedings of the 5th International Workshop on Document Analysis Systems V*, pages 188–199, London, UK, 2002. Springer-Verlag.

[17] R. Brugger, A. Zramdini, and R. Ingold. Modeling documents for structure recognition using generalized n-grams. In *ICDAR'97: Proceedings of the 4th International Conference on Document Analysis and Recognition*, pages 56–60, Washington, DC, USA, 1997. IEEE Computer Society.

[18] H. Chao and J. Fan. Layout and content extraction for pdf documents. In *IAPR International Workshop on Document Analysis Systems (DAS'04)*, pages 213–224, Florence, Italy, 2004.

[19] H. Chao and L. Xiaofan. Capturing the layout of electronic documents for reuse in variable data. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 940–944, Seoul, Korea, 2005.

[20] L. Cinque, L. Lombardi, and G. Manzini. A multiresolution approach for page segmentation. *Pattern Recognition Letter*, 19(2):217–225, 1998.

[21] ABC Amber PDF Converter, 2009. http://www.processtext.com/.

[22] PDF-File Converter, 2009. http://www.pdf-file.com.

[23] PDF XML Converter, 2009. http://www.pdf2text.com.

[24] PDFText Converter, 2009. http://www.filehunter.com.

[25] Smart PDF Converter. http://www.pdftodocconverterpro.com.

[26] Easy PDF Creator, 2009. http://www.pdfdesk.com.

[27] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

[28] H. Déjean and J.-L. Meunier. A system for converting pdf documents into structured xml format. In *IAPR International Workshop on Document Analysis Systems (DAS'06)*, pages 129–140, Nelson, New Zealand, 2006.

[29] D.S. Doermann and R. Furuta. Image based typographic analysis of documents. In *Proceedings of the Second International Conference on Document Analysis and Recognition, ICDAR 1993*, pages 769–773, October 1993.

[30] D. Dori, D. Doermann, C. Shin, R. Haralick, I. Philips, M. Buchman, and D. Ross. *Handbook on Optical Character Recognition and Document Image Analysis*, chapter The Representation of Document Structure: A Generic Object-Process Analysis, pages 421–456. World Scientific, 1997.

[31] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2000.

[32] INfix PDF Editor, 2009. http://www.iceni.com.

[33] F. Esposito, D. Malerba, and G. Semeraro. Automated acquisition of rules for document understanding. In *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, pages 650–654, October 1993.

[34] K.-C. Fan, C.-H. Liu, and Y.-K. Wang. Segmentation and classification of mixed text/graphics/image documents. *Pattern Recogn. Lett.*, 15(12):1201–1209, 1994.

[35] Jon Ferraiolo and ed. Scalable vector graphics (svg) 1.0 specification, 2001.

[36] J. Fisher, S. Hinds, and D. D'Amato. A rule-based system for document image segmentation. *10th International Conference on Pattern Recognition*, pages 567–570, 1990.

[37] Achille Fokoue, Kristoffer Rose, Jérôme Siméon, and Lionel Villard. Compiling xslt 2.0 into xquery 1.0. In *Proceedings of the 14th international conference on World Wide Web*, pages 682–691, New York, NY, USA, 2005. ACM.

[38] R. P. Futrelle, M. Shao, C. Cieslik, and A. E. Grimes. Extraction, layout analysis and classification of diagrams in pdf documents. In *Seventh International Conference on Document Analysis and Recognition (ICDAR'03)*, pages 1007–1012, Edinburgh, Scotland, 2003.

[39] B. Gatos, S.L. Mantzaris, K.V. Chandrinos, A. Tsigris, and S.J. Perantonis. Integrated algorithms for newspaper page decomposition and article tracking. *Document Analysis and Recognition, International Conference on*, 0:559, 1999.

[40] GetPDF, 2009. http://www.getpdf.com.

[41] V. Govindaraju, S. W. Lam, D. Niyogi, D. B. Sher, R. K. Srihari, S. N. Srihari, and D. Wang. Newspaper image understanding. In *KBCS '89: Proceedings of the International Conference on Knowledge Based Computer Systems*, pages 375–384, London, UK, 1990. Springer-Verlag.

[42] PDF Grabber, 2009. http://www.pixelplanet.com.

[43] K. Hadjar, M. Rigamonti, D. Lalanne, and R. Ingold. Xed: A new tool for extracting hidden structures from electronic documents. In *DIAL'04: Proceedings of the First International Workshop on Document Image Analysis for Libraries (DIAL'04)*, page 212, Washington, DC, USA, 2004. IEEE Computer Society.

[44] Patrick Haffner, Paul G. Howard, Patrice Simard, Yoshua Bengio, and Yann Lecun. High quality document image compression with djvu. *Journal of Electronic Imaging*, 7:410–425, 1998.

[45] M. R. B. Hardy, D. Brailsford, and P. L. Thomas. Creating structured pdf files using xml templates. In *ACM Symposium on Document Engineering (DocEng'04)*, pages 99–108, Milwaukee, USA, 2004.

[46] Elliote Rusty Harold and W. Scott Means. *XML in a nutshell*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[47] P. Héroux, E. Trupin, and Y. Lecourtier. Modélisation et classification pour la rétroconversion des documents. In *CIFED Colloque International Francophone sur l'Ecrit et le Document*, pages 413–421, Lyon, France, July 2000.

[48] O. Hitz. *A Framework for Interactive Document Recognition*. PhD thesis, PhD Thesis, IIUF-Universit de Fribourg, Switzerland, 2005. thesis Nr. 1488.

[49] O. Hitz, L. Robadey, and R. Ingold. An architecture for editing document recognition results using xml. In *4th IAPR International Workshop on Document Analysis Systems (DAS'2000)*, pages 385–396, Rio de Janeiro, Brazil, December 2000.

[50] W. Horak. Office document architecture and office document interchange formats: Current status of international standardization. *Computer*, 18(10):50–60, 1985.

[51] T. Hu. *New Methods for Robust and Efficient Recognition of the Logical Structures in Documents*. PhD thesis, PhD Thesis, IIUF-Université de Fribourg, 1994.

[52] ScanSoft Inc. Xdoc data format: Technical specification version 4.0, May 1999.

[53] Adobe Systems Incorporated. Pdf reference, sixth edition: Adobe portable document format version 1.7, November 2006. http://www.adobe.com/devnet/pdf/pdf_reference.html.

[54] Adobe Systems Incorporated. Acrobat reader, 2009. http://www.adobe.com/fr/products/acrobat.

[55] Adobe Systems Incorporated. Portable document format (pdf): A history of trust, 2009. http://www.adobe.com/pdf/about/history.

[56] Y. Ishitani. Logical structure analysis of document based on emergent computation. *IEICE Transactions on Information and Systems*, E88-D(8):1831–1842, 2005.

[57] JPedal, 2009. http://www.jpedal.org.

[58] K. Kise, A. Sato, and M. Iwata. Segmentation of page images using the area voronoi diagram. *Computer Vision and Image Understanding*, 70(3):370–382, 1998.

[59] PDF Export Kit, 2009. http://www.pdfkit.com.

[60] S. Klink, A. Dengel, and T. Kieninger. Document structure analysis based on layout and textual features. In *Proc. of International Workshop on Document Analysis Systems, DAS2000*, pages 99–111. IAPR, 2000.

[61] M. Krishnamoorthy, G. Nagy, S. Seth, and M. Viswanathan. Syntactic segmentation and labeling of digitized pages from technical journals. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 15(7):737–747, 1993.

[62] S.W. Lam, D. Wang, and S.N. Srihari. Reading newspaper text. In *Pattern Recognition, 1990. Proceedings., 10th International Conference on*, volume i, pages 703–705 vol.1, Jun 1990.

[63] L. Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, 2nd edition edition, July 1994.

[64] F. Le Bourgeois, H. Emptoz, and S.S. Bensafi. Document understanding using probabilistic relaxation: Application on tables of contents of periodicals. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 508–512, 2001.

[65] F. LeBourgeois and S. Souafi-Bensafi. Rasade: Automatic recognition of structured document using typography and spatial inference. In *Document Layout Interpretation Analysis (DLIA'09)*, Bangalore, India, 1999.

[66] K.H. Lee, Y.C. Choy, and S.B. Cho. Logical structure analysis and generation for structured documents: A syntactic approach. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1277–1294, September/October 2003.

[67] P. Lefevre and F. Reynaud. Odil: an sgml description language of the layout structure of documents. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 480–488 vol.1, Aug 1995.

[68] T. L. Lenox and C. R. Woratschek. Optical character recognition. *The Gale Group Inc. 2002. Encyclopedia.com, Computer Sciences*, March 2009.

[69] W. S. Lovegrove and D. F. Brailsford. Document analysis of pdf files: methods, results and implications. *Electronic Publishing - Origination, Dissemination and Design*, 8(3):207–220, 1995.

[70] S. Mao, A. Rosenfeld, and T. Kanungo. Document structure analysis algorithms: A literature survey. In *ITCC International Conference on Information Technology: Coding and Computing*, pages 197–207, Maryland, USA, 2003.

[71] N. Marovac. Page description languages: concepts and implementations. *Workstations and Publication Systems*, pages 14–26, 1987.

[72] T. McKinley. Why pdf is everywhere. *Inform, the journal of AIIM*, 11(8), 1997.

[73] C. Musciano and B. Kennedy. *HTML & XHTML: The Definitive Guide*. O'Reilly, sixth edition edition, October 2006.

[74] G. Nagy. Twenty years of document image analysis in pami. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):38–62, 2000.

[75] George Nagy, Sharad Seth, and Mahesh Viswanathan. A prototype document image analysis system for technical journals. *Computer*, 25(7):10–22, 1992.

[76] W. Ng, W.-Y. Lam, and J. Cheng. Comparative analysis of xml compression technologies. *World Wide Web*, 9(1):5–33, 2006.

[77] D. Niyogi and S. N. Srihari. Knowledge-based derivation of document logical structure. In *in Proceedings of International Conference on Document Analysis and Recognition*, pages 472–475, 1995.

[78] A. L. Oakley and A. C. Norris. Page description languages: development, implementation and standardization. *Electron. Publ. Origin. Dissem. Des.*, 1(2):79–96, 1988.

[79] L. O'Gorman. The document spectrum for page layout analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1162–1173, 1993.

[80] Lawrence O'Gorman and Rangachar Kasturi. *Document image analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[81] M. D. Paknad and R. M. Ayers. Method and apparatus for identifying words described in a portable electronic document. U.S. Patent 5,832,530, 1998.

[82] G. I. S. Palmero, J. M. C. Izquierdo, Y. A. Dimitriadis, and J. L. Coronado. A new neuro-fuzzy system for logical labeling of documents. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 4, pages 431–435, Aug 1996.

[83] T. Pavlidis and J. Zhou. Page segmentation and classification. *CVGIP Graphical Models and Image Processing*, 54(6):484–496, 1992.

[84] Nitro PDF, 2009. http://www.nitropdf.com.

[85] Planet PDF, 2009. http://www.planetpdf.com.

[86] Solidconverter PDF, 2009. http://www.archisoftint.com.

[87] PDF-Convert, 2009. http://www.pdf-convert.com.

[88] PDF2Office, 2009. http://www.recosoft.com.

[89] PDFText, 2009. http://www.pdf-analyzer.com.

[90] PDFTron, 2009. http://www.pdftron.com.

[91] Lynne A. Price. Practical sgml as an introduction to sgml. *SIGDOC Asterisk J. Comput. Doc.*, 20(2):36–38, 1996.

[92] DocSmartz Pro, 2009. http://www.docsmartz.com.

[93] Inc. RAF Technology. Dafs: Document attribute format specification, 1995.

[94] F. Rahman and H. Alam. Conversion of pdf documents into html: a case study of document image analysis. In *Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers 2003*, pages 87–91, USA, 2003.

[95] Y. Rangoni and A. Belaïd. Document logical structure analysis based on perceptive cycles. In *IAPR International Workshop on Document Analysis Systems (DAS06)*, pages 117–128, Nelson, New Zealand, 2006. Springer Berlin / Heidelberg.

[96] M. Rigamonti, K. Hadjar, D. Lalanne, and R. Ingold. Xed: un outil pour l'extraction et l'analyse de documents pdf. In *In proc. of Huitième Colloque International Francophone sur l'Ecrit et le Document (CIFED'04)*, pages 85–90, La Rochelle, France, 2004.

[97] L. Robadey. *2(CREM) : Une méthode de reconnaissance structurelle de documents complexes basée sur des patterns bidimensionnels.* PhD thesis, PhD Thesis, IIUF-Université de Fribourg, Switzerland, 2001.

[98] Azriel Rosenfeld and Avinash C. Kak. *Digital Picture Processing.* Academic Press, Inc., Orlando, FL, USA, 1982.

[99] R. Smith. Hybrid page layout analysis via tab-stop detection. In *ICDAR International Conference on Document Analysis and Recognition*, Barcelona, Spain, 2009.

[100] A.L. Spitz. Style-directed document segmentation. In *Symposium on Document Image Understanding Technology*, pages 195–199, Columbia, USA, 2001.

[101] B. Stehno and G. Retti. Modelling the logical structure of books and journals using augmented transition network grammar. *Journal of Documentation*, 59:69–83, 2003.

[102] Mimotek Structuriser, 2009. http://www.mimotek.com.

[103] PDF Studio, 2009. http://www.qoppa.com.

[104] K.J. Swanson and J. Judt. Comprex: Further developments in xml compression. In *Aerospace Conference, 2006 IEEE*, pages 1–7, 0-0 2006.

[105] TEXTfromPDF, 2009. http://www.textfrompdf.com.

[106] VeryPDF Tools, 2009. http://www.verypdf.com.

[107] PDF Transformer, 2009. http://www.pdftransformer.com.

[108] Trapeze, 2009. http://www.mesadynamics.com.

[109] S. Tsujimoto and H. Asada. Understanding multi-articled documents. In *Pattern Recognition, 1990. Proceedings., 10th International Conference on*, volume 1, pages 551–556, June 1990.

[110] H. Walischewski. Automatic knowledge acquisition for spatial document interpretation. *Document Analysis and Recognition, International Conference on*, 0:243, 1997.

[111] N. Walsh and L. Muellner. *DocBook: The Definitive Guide.* O'Reilly, 1999.

[112] Dacheng Wang and Sargur N. Srihari. Classification of newspaper image blocks using texture analysis. *Computer Vision Graphics and Image Processing*, 47(3):327–352, 1989.

[113] Wikipedia. Device independent file format — wikipedia, the free encyclopedia, 2009. [Online; accessed 8-January-2010].

[114] Wikipedia. Portable document format — wikipedia, the free encyclopedia, 2009. [Online; accessed 17-August-2009].

[115] Wikipedia. Rich text format — wikipedia, the free encyclopedia, 2009. [Online; accessed 8-January-2010].

[116] Wikipedia. Scrgb — wikipedia, the free encyclopedia, 2009. [Online; accessed 18-January-2010].

[117] Wikipedia. Xml — wikipedia, the free encyclopedia, 2009. [Online; accessed 5-September-2009].

[118] Wikipedia. Microsoft word — wikipedia, the free encyclopedia, 2010. [Online; accessed 5-January-2010].

[119] Wikipedia. Open xml paper specification — wikipedia, the free encyclopedia, 2010. [Online; accessed 8-January-2010].

[120] K. Y. Wong, R. G. Casey, and M. Wahl. Document analysis system. *IBM journal of Research and Development*, 26(6):647–656, November 1982.

[121] PDF 2 Word, 2009. http://www.cadkas.de.

[122] Xpdf, 2009. http://www.glyphandcog.com.

[123] xtPDF Extractor, 2009. http://www.exti.com.

[124] S. Yacoub, J. Burns, P. Faraboschi, D. Ortega, J. A. Peiro, and V. Saxena. Document digitization lifecycle for complex magazine collection. In *DocEng '05: Proceedings of the 2005 ACM symposium on Document Engineering*, pages 197–206, New York, NY, USA, 2005. ACM.

[125] M. Yamaoka, O. Iwaki, N. Babaguchi, and T. Kitahashi. Interactive approach to the extraction of logical structures from unformatted document images using a substructure model. *Document Analysis and Recognition, International Conference on*, 0:185, 1999.

List of Figures

List of Tables

Curriculum Vitæ

Jean-Luc Bloechle

Personal Information

Date of Birth	April 19, 1977
Nationality	Swiss
Languages	French (mother tongue), English (fluent), German (scholar knowlegde)

Education

2005 - 2010	PhD in Computer Science Faculty of Science, University of Fribourg, Switzerland
2004 - 2008	Diploma in Higher Education and Technology of Education Centre de Didactique Universitaire, University of Fribourg, Switzerland
2002 - 2004	Master of Science in Computer Science Faculty of Science, University of Fribourg, Switzerland
1998 - 2002	Bachelor of Science in Computer Science Including a minor in "Sociologie de la communication et des médias" Faculty of Science, University of Fribourg, Switzerland
1997 - 1998	Passed foundation course for first-year university in Physics Faculty of Science, University of Fribourg, Switzerland
1992 - 1997	High-school diploma in Sciences (Maturité fédérale) Collège St-Michel, Fribourg, Switzerland
1996 - 1997	High-school award: "Prix de l'amicale des physiciens de la protection AC" Collège St-Michel, Fribourg, Switzerland
1994 - 1995	High-school award: "Prix Friedrich Dessauer: meilleur élève en sciences" Collège St-Michel, Fribourg, Switzerland

Research Interests

- Image processing & pattern recognition
- Document engineering & understanding
- Physical and logical structure recognition of documents
- Development of XML & Java technologies related to PDF: XED - Dolores - OCD

List of Publications

1. J.-L. Bloechle and R. Ingold, *Restructuration physique et logique de documents électroniques textuels*, 12e Colloque International sur le Document Electronique (CiDE.12), Montréal, Canada, 21-23 octobre, 2009, pp. 287-297.

2. J.-L. Bloechle, D. Lalanne and R. Ingold, *OCD: An Optimized and Canonical Document Format*, 9th International Conference on Document Analysis and Recognition (ICDAR'09), Barcelona, Spain, July 22-29, 2009, pp. 236-240.

3. M. Baechler, J.-L. Bloechle, A. Humm, R. Ingold and J. Hennebert, *Labeled Images Verification Using Gaussian Mixture Models*, 24th Annual ACM Symposium on Applied Computing (ACM SAC'09), Honolulu, Hawaii, USA, March 8-12, 2009, pp. 1331-1336.

4. J.-L. Bloechle, C. Pugin and R. Ingold, *Dolores: An Interactive and Class-Free Approach for Document Logical Restructuring*, 8th IAPR Workshop on Document Analysis Systems (DAS'08), Nara, Japan, September 16-19, 2008, pp. 644-652

5. J.-L. Bloechle, M. Rigamonti, D. Lalanne and R. Ingold, *XCDF: Un format canonique pour la représentation de documents*, Colloque International Francophone sur l'Ecrit et le Document (CIFED'06), Fribourg, Switzerland, September 18-22, 2006, pp. 19-23.

6. J.-L. Bloechle, M. Rigamonti, K. Hadjar, D. Lalanne and R. Ingold, *XCDF: A Canonical and Structured Document Format*, 7th IAPR Workshop on Document Analysis Systems (DAS'06), Nelson, New Zealand, February 13-15, 2006, pp. 141-152

7. M. Rigamonti, J.-L. Bloechle, K. Hadjar, D. Lalanne and R.Ingold, *Towards a Canonical and Structured Representation of PDF Documents through Reverse Engineering*, 8th International Conference on Document Analysis and Recognition (ICDAR'05), Seoul, Korea, August 29 - September 01, 2005, pp. 1050-1054.

8. J.-L. Bloechle, *Réseau de neurones artificiels pour la classification des fontes arabes et la distinction entre la langue arabe et les langues latines*, Master thesis, Internal Publication, University of Fribourg, June 2003.