A System for Converting PDF Documents into Structured XML Format

Conference Paper · February 2006

DOI: 10.1007/11669487_12 · Source: DBLP

2 authors, including:

Jean-Luc Meunier
Naver Labs Europe
57 PUBLICATIONS **582** CITATIONS

SEE PROFILE

A System for Converting PDF Documents into Structured XML format

Hervé Déjean, Jean-Luc Meunier[1]

Xerox Research Centre Europe
6, chemin de Maupertuis
F-38240 Meylan
Firstname.Lastname@xrce.xerox.com

Abstract. We present in this paper a system for converting PDF legacy documents into structured XML format. This conversion system first extracts the different streams contained in PDF files (text, bitmap and vectorial images) and then applies different components in order to express in XML the logically structured documents. Some of these components are traditional in Document Analysis, other more specific to PDF. We also present a graphical user interface in order to check, correct and validate the analysis of the components. We eventually report on two real user cases where this system was applied on.

1 Introduction

Enterprise Content Management (ECM) software enables organizations to create/capture, manage/secure, store/retain/destroy, publish/distribute, search, personalize, and present/view/print any digital content [1]. The capture module essentially offers functionalities such as scanning, OCR, indexing. Most of the ECM systems integrate now XML, a format which provides, among other advantages, a way to store documents with metadata and structural information. Due to the dissemination of this format and its adoption as standard, new requests from organizations challenge ECM software to provide more sophisticated functionalities such as document conversion to XML which keeps the structural information. This structural information is not explicitly marked up in most electronic documents. This is particularly true for PDF documents [2]. Even if the version 6 of PDF allows a user to create a file containing structural information, most of them do no contain such information. The use of PDF format as input format for a conversion task can be questionable: since PDF files are very often generated from another format (MS Word, Latex), converting files with the original format could be more efficient. But the everyday life (and customers cases provided by Xerox business units) shows that PDF is now a common exchange format between organizations, and often is the only accessible format.

[1] This work is supported by VIKEF Integrated Project co-funded under the EU 6th Framework Programme.

Portable Document Format (PDF) goal is "to enable users to exchange and view electronic documents easily and reliably, independently of the environment in which they were created" [2]. A detailed presentation of this format is out of the scope of this article, and we will consider here a PDF document as a sequence of pages, each page being composed of any combination of text (referred as text objects in [2]), graphics (path objects) and images (external objects).

1. a text object consists in one or more characters and layout information (position, fonts)
2. a path object contains vectorial instructions (lines or bezier curves).
3. a image external object defines a rectangular image.

Many PDF converters [3;4;5] are available off the shelf, but often they simply provide a format conversion, and almost no structural information is provided in the final format, except at the very low level (words, lines). Information contained in the original PDF file is in a way simply translated in another format. [6] presents a comparison of some of them.

OCR-oriented software [7;8;9] now provides a functionality to convert PDF through scanning and OCR. The drawback is that information clearly present in the PDF file (text zone, external image) is not used, and the image analysis step can introduce noise (some text may not be recognized as text zone; some images may not be correctly recognized). Nevertheless, this strategy is the only one available when the PDF file only contains images (from scanning).

[6] proposes a method combining both approaches: applying traditional layout analysis on TIFF images generated from PDF combined with low-level content extracted from the PDF file. We present here a system that relies solely on the PDF-extracted content, no longer requiring the conversion to TIFF nor the combination of both approaches. The first steps consist in extracting the native PDF pieces of information: text, path objects, and external objects (Sections 2,3). Then the textual organization of each page and its reading order is computed through a XY-cut-based algorithm (Sections 4,5). In order to extract the logical structure at the document level, we first detect the table of contents (Toc) of the document, and structure it according to the hierarchical information present in its Toc (Section 6). Section 5 and 6 are brief since this work has been reported in [14;18]. We also present a graphical user interface which allows the user to correct each step of the conversion (if necessary). We eventually discuss two user cases where this conversion system has been successfully used.

2 Text Extraction

If PDF allows the preservation of the look of documents, it does not contain or guarantee a correct logical representation of the text. An inspection of the text streams extracted form PDF files reveals first that these streams can correspond to various objects: a character, a partial word, a word, a line,…. Secondly the order of these text streams does not always correspond to the reading order. A word reconstruction component and a reading order component are then necessary in order to correctly

extract the text from a PDF file. We will describe here the word reconstruction component we have developed. The reading order component is explained Section 5.

Text objects are extracted from the PDF and a word and line segmentation is produced based on heuristics using the distance between characters and their geometrical positions (similarly to the heuristics present in the Xpdf library [10]). In most cases, this set of heuristics correctly segments the text streams into words and lines. Table 1 shows tokenization some errors provided by this method.

Table 1. Texts before and after correction

Original text	Corrected text
TI GH TEN I N G TO RQ UES	TIGHTENING TORQUES
Throttle p edal position sensor	Throttle pedal position sensor
F it t in g r ear s eal	Fitting rear seal
Oil vap ou r f u ll r ec ir c u lat ion s ys t em (B low -b y)	Oil vapour full recirculation system (Blow - b y)
En gine coolant temperatu re sensor	Clutch pedal position sensor
Brake pedal p osition senso r	Brake pedal position sensor
Ch ec kin g t o r s ion	Checking torsion
E N G I N E E R I N G	E N G I N E E R I N G
Ch ec kin g b en d in g	Checking bend ing
Ch ec kin g t o r s ion	Checking torsion
Ru n u p	Run up

Even if the errors are marginal, texts which are wrongly tokenized often correspond to logical elements which structure documents, such as document/chapter/section headings. One possible explanation is that special fonts and layout are used for these elements. It is noteworthy that extracting these elements correctly might be very important since they structure the document (as the reader will see Section 6). To correct these errors, we propose the following method:

1. text is extracted from PDF using geometrical information.
2. a weighted lexicon is built, based on the tokens present in the document
3. for each line, all the possible tokenizations are generated
4. the best tokenization is selected

After the first extraction using geometrical information, a lexicon is built. Since most of the word segmentation is correctly done, the lexicon mainly contains correct words. The hypothesis we do here is that words which occur in ill-formed elements will occur correctly in other parts of the document. We will then use correct tokenized words in order to retokenize bad words. External lexicons can be of course used but, since document conversion mainly involves technical documents (with a domain-specific terminology), the general lexicons may have a bad coverage.

Each token (word of the document) is associated a weight. The weighting schema we use is the following one:

$$W = \text{length}(\text{token}) * \log(\text{frequency}(\text{token}) + 1) \quad (1)$$

where frequency(token) corresponds to the number of token occurrences in the document, and length(token) is the length in character. We consider that the more frequent a word, the more reliable.

A weighted automaton (see [11]) is built, describing the set of all tokens of the document with their respective weight. Let us call it D (for dictionary).

In order to generate all the possible tokenizations, we apply (through a transducer called T) the following actions over an input string (which corresponds to a line built step 1): a space character is deleted or a space character is inserted.

We associate a weight to both operations. Since most of the errors are corrected by deleting a space character, we give a higher weight to the deletion operation. An automaton is generated for a string we want to re-tokenize such that each letter of the string labels a transition. Let us call this automaton S. S, T and D are composed in order to generate all the tokenization for S: S .o. T .o. D* (* being the kleen star). This final transducer will assign to each possible tokenization a weight using the weights associated to the words (automaton D) and to the operation (transducer T). A Viterbi algorithm is applied over the transducer in order to select the path with the highest weight.

Table 1 shows texts extracted from a PDF file and the new tokenization after our correction component. Tests have been made with correctly tokenized text, and the correct tokenization is kept unchanged by the method. This method also provides dealing with hyphenation. By simply adding a third operation: deleting the hyphen symbol, the resulting automaton will de-hyphenate hyphenated words which occur in the document somewhere else (or in an external lexicon).

3 Image/Text Separation

PDF contains also information about images, mainly using two objects: external objects which allow the insertion of external objects (as raster images), and path objects which allow the description of vectorial elements (the clipping objects are not yet taken into account by our system). A single vectorial image can be composed of thousands of paths. One problem is then to regroup all the elements (paths, text, images) which form the complete image. This is a traditional task for Document Analysis systems to recognize and label parts of pages as text zones and image zones. If these methods can be also used here, we prefer to use the information present in the PDF file. An XY-cut-based algorithm (Section 5) is first applied on external and path objects, first ignoring text. The zone of a path object is defined thanks to the coordinates of each element of the path (an approximation is in practice enough for bezier curves). Ignoring textual elements allows avoiding many situations where images and text zones form non-manhattan zones, a well-known problem for an XY-cut approach. The graphical elements (path objects and images) are then grouped, and the final groups can contain both: images and paths. Text segmentation is explained Section 5. Once elements have been regrouped into zones, the labeling (text/image zone) is done by computing the surface of each type (text, image) in the zone. Since overlap between a text and an image zones can happened, both are merged, and the label of this zone corresponds to the label of the text or image zone which covers the highest surface. This simple approach provides a robust solution in most cases, but fails in the following cases:

1. the page contains a background image: The following heuristic can be used in this case: when an image has a size which is similar to the page size, consider it as background image.

2. an image/schema is composed of different elements which are too distant: in this case the image is oversegmented
3. Text is wrongly integrated into an image zone. This error typically occurs when the author wants to insert an image onto a page, although space is clearly missing.

This component is still under research but the current status is enough in order to correctly detect vectorial images, and consequently can be used for developing other components such as a caption detector.

4 Header Footer Detection

The purpose of this component is to detect zones at the top of the document and at the bottom of the document which correspond to headers and footers. Since one goal of this system is to logically structure documents, the deletion of pages induces the deletions of elements which are directly linked to the page segmentation, typically headers and footers. In order to characterize this zone, we use the following observation: in a header or footer zone, the textual variety is much lower than in the body page (see Table 3). A similar observation is used in [12]. The header/footer detection consists of three main steps: text normalization, textual variability computation and header/footer zone detection

The only normalization applied (step 1) consists in replacing all digits by a unique character (D). This normalization is due to the frequent use of page/chapter/section numbering and dates in headers/footers.

The text extraction component provides us with the vertical or horizontal positions of textual fragments in a page. To each vertical position, we associated the number of text blocks occurring at this position, and the number of different text blocks occurring at this position. A textual variability score for each position i is computed as follows:

$$tv\_score(i) = \frac{\#different\ \ text\ \ blocks}{\#total\ \ text\ \ blocks} \qquad (1)$$

For example, in the Linux System Administrator's Guide, the texts occurring at the position 108 have a textual variability of 7/93 =0.075 (Table 2). As Table 3 shows, and accordingly with our hypothesis, the position 108 has a very low score (header), while the position 156 (page body) has a very high one.

Table 2. Variability for the position 108 in the Linux System Administrator's Guide

Text at the position y = 108	Nb of occurrences
Installing and Configuring DD/DDDBase-T X/DDDD	25
DD/DDDBase-TX Interface Card Statistics	9
Troubleshooting DD/DDDBase-TX/DDDD	33
Configuring Network Connectivity Using SAM'	5
DDDBase-TX Resources	9
Hardware Regulatory Statements	3
Hardware Reference Information	9
total	93

Table 3. Textual variability according to the position in a page: The Linux System Administrator's Guide

position	# text blocks	#different text blocks	Textual variability
96	5	3	0.6
108	93	7	0.075
122	89	32	0.35
143	1	1	1
155	17	11	0.67
156	47	45	0.96

Once the variability score for each position is computed, we try to identify whether the document has header zones and footer zones. For this, we use the following method:

1. Identify potential headers/footers elements: all elements with a score lower than a given threshold θ (0.5 in practice) are identified as potential headers or footers. The top first potential candidate is identified (starting from the top for the header and from the bottom for the footer). For the header detection, this candidate must occur in the upper half-page, for the footer detection, the candidate must occur in the lower half-page.
2. Merge surrounding elements: We extend the current header zone (resp. footer list) with preceding and following elements if and only if its insertion decreases the textual variability score of the new augmented list. Elements are added incrementally starting from the adjacent elements of the list. Potentially no new element is added. The reduction of the score imposes that the new element has some text in common with the current list. If the final zone does not reach the top (for header) or bottom (for footer) of the page, the zone is invalidated.
3. Return the lowest (resp. highest) position of the header (resp. footer) list. A zone for header and/or a zone for footer are then recognized, and elements occurring in them are considered as header or footer.

This method works very well when headers and footers are homogenous over the entire document. It can partially fail when a document is composed of parts which have different kinds of headers or footers (headers in an annex can be different (e.g. lower) to headers in the document body).

5 Reading Order Computation and Segmentation into Paragraphs

The ordering problem consists in ordering the objects of a page in order to reflect the human reading order. Multiple approaches to this problem have been proposed in the literature [13], exploiting geometric or typographic features of the page objects, or going further in exploiting the content of objects, with or without priori knowledge about a particular document class.

The use of off-the-shelf PDF converters leads to the consideration of layout objects of various granularities, because they may contain one line, or one word, or part of a word, or even a single letter. The pages considered here often contain several hundred

of textual objects, and we therefore proposed in [14] a method based on the XY-Cut [15], which takes an optimization approach to the problem and leverages dynamic programming to process efficiently any page.

It is also often important to segment the flow of text into paragraphs. The XY-cut approach can also provide this segmentation when the line spacing indicates the paragraph boundaries.

We evaluated the method thanks to the UW III document image database, which includes 1600 English journal pages. We tested at both the word-level and line-level using the ground truth. The parameters were set to their default values (only taking into account the image resolution factor, since the scale was 4 times larger than with our PDF converters). We observed an error of less than 1% of misplaced objects.

The method is a pure geometric ordering method, of general applicability in the sense that no domain knowledge is used. But there exists geometrically ambiguous pages (rare in our experience with technical documents) and for those, additional features must be taken into account. Fortunately, the score function offers convenient room for improvements with the same approach. This method suffers from the XY-Cut L-Shapes weakness but is fast (20 pages/second on a Pentium 4) thanks to the dynamic programming technique. We are now interested in exploring alternative methods based on 2D relationships [16;17].

6 Toc-Based Structuring

This module aims at structuring a document according to its table of contents (hereafter ToC). First the toc of a document is automatically detected, and in the same step, the ToC entries are linked to their entries in the document body. Finally the hierarchical structure of the toc is used to structure the document accordingly. A more detailed presentation of the method is presented in [18].

In view of the large variation in shape and content a ToC may display, we believe that a descriptive approach would be limited to a series of specific collections. Therefore, we instead chose a functional approach that relies on the functional properties that a ToC intrinsically respects. These properties are:

1. Contiguity: a ToC consists of a series of contiguous references to some other parts of the document itself;
2. Textual similarity: the reference itself and the part referred to share some level of textual similarity;
3. Ordering: the references and the referred parts appear in the same order in the document;
4. Optional elements: a ToC entry may include (a few) elements whose role is not to refer to any other part of the document, e.g. decorative text;

Our hypothesis is that those 4 properties are sufficient for the entire characterization of a ToC, independently of the document class and language.

6.1 The Table of Contents Detection

Three steps permit us to identify the area of the document containing the ToC text.

Firstly, links are defined between each pair of text blocks in the whole document satisfying a textual similarity criterion. Each link includes a source text block and a target text block. The similarity measure we currently use is the ratio of words shared by the two blocks, considering spaces and punctuation as word separators. Whenever the ratio is above a predefined threshold, the similarity threshold, a pair of symmetric links is created. In practice, 0.5 is a good threshold value to tolerate textual variation between the ToC and the document body while avoiding too many noisy links.

Secondly, all possible ToC candidate areas are enumerated. A brute force approach works fine. It consists in testing each text block as a possible ToC start and extending this ToC candidate further in the document until it is no longer possible to comply with the five properties identified above. A ToC candidate is then a set of contiguous text blocks, from which it is possible to select one link per block so as to provide an ascending order for the target text blocks.

Thirdly, we employ a scoring function to rank the candidate tables of contents. The highest ranked candidate is then selected for further processing. Currently, the scoring function is the sum of entry weights, where an entry weight is inversely proportional to the number of outgoing links. This entry weight characterizes the certainty of any of its associated links, under the assumption that the more links initiate from a given source text block, the less likely that any one of those links is a "true" link of a table of contents.

Once the highest ranked table of contents candidate has been selected, we select the best link for each of its entries by finding a global optimum for the table of contents while respecting the five ToC properties. A weight is associated with each link, which is proportional to the similarity level that led to the link creation. A Viterbi shortest-path algorithm is adequate to effectively determine the global optimum.

6.2 Hierarchical structuring using the ToC Hierarchy

The next step is then to find the hierarchical organization of the ToC. This is done in a twofold process:

Entry clustering: the ToC entries are clustered according to their visual characteristics (fonts, positions, case). The assumption made here is to consider that elements belonging to the same hierarchical level share the same visual characteristics. This clustering is done using state-of-the-art algorithms. The output is a set of clusters which correspond to each hierarchical level. If the document hierarchy is not reflected by visual clues, then no hierarchy will be found.

Cluster hierarchy determination: The purpose of this second step is to find the hierarchical relation between clusters. Acknowledging the fact that the first element of the ToC is not necessarily an element belonging to the highest level (a document with a complex front matter for example), we use the following heuristics: the elements (namely ToC Entries) of lowest level more frequently have adjacent elements from the same level as elements of higher levels. For instance, for a document with three hierarchical levels (chapter, section, subsection), subsections very often have at least one adjacent element of the same level (if only ToC entries are taken into account). This heuristics allows us to identify the lowest hierarchical

level. The procedure is iterated by ignoring the cluster just identified. The detailed procedure is given in [18].

The result of this procedure is the hierarchical structuring of the ToC entries. The ultimate step simply consists in first marking each heading in the document body with its hierarchical level, and then in structuring the document accordingly. The final document is now segmented in hierarchical sections similarly to the information present in the ToC.

7 The Graphical User Interface

We developed a graphical tool to allow the operator to set up the whole conversion chain for a given collection and validate and/or correct the processing output. The conversion chain set up consists in determining which components to apply and with which parameterization, while the correction/validation activity consists in verifying the quality of the conversion and correcting it if needed. Actually, both activities are interleaved since the setting up involves verifying the output while tuning the settings.

Figure 2 shows a screenshot of the graphical user interface of the tool. User-centered design involving cycles of task analysis, mock-ups and user testing allowed us to design a tool taking into account the main needs:

- Handling of large collections of documents;
- Fast display of any large document, with instantaneous page browsing and fast document loading (less than 5s for a 1000 page document);
- Capability of customizing both the conversion process and the settings of each conversion step;
- Capability to explore the concurrently possible alternate settings;
- Visual rendering of the conversion output thanks to an intuitive graphical page decoration overlaid;
- An XML display with extensible perspectives offering appropriately customized views, thanks for instance to a series of dedicated XSLT transforms or to dedicated applicative code.
- In-place correction mechanism acting on the overlaid decorations with consistency control;
- A plug-in mechanism to embed future components;

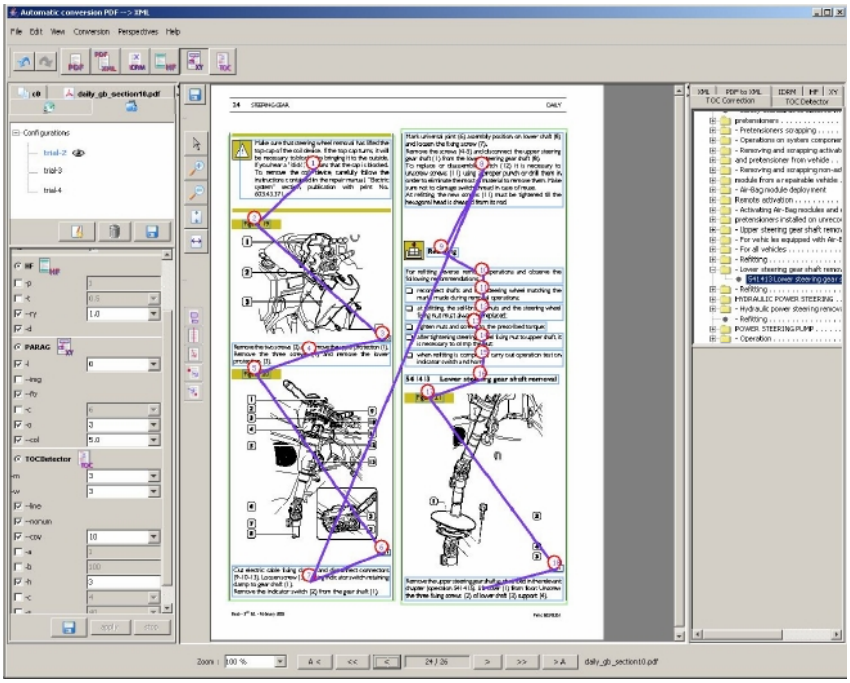

Fig. 1. Left side: the collection, the open documents, the tested configurations. Right side: different perspectives dedicated to each conversion component (here the Toc detector). Center part: decoration for segmentation in blocks and the reading order. In the toolbar appear the different possible main views corresponding to each conversion steps

8 Final XML Output

For lack of space, we only present the main element of our final format: documents are now organized in blocks, and no longer in pages. Blocks have some similarities with the compound texton presented in [19], consisting of a header, body, and optional trailer. Our block consists of an optional head, a body and an optional tail. Each of these three parts contains either textual data or a block. Blocks are then recursive, similarly to the compound texton. This generic schema allows us to capture the most frequent structures found in documents.

Two User Cases

We present here two customer cases where this system was applied on: conversion of car repairing manuals and construction contracts. In the two cases, the purpose of this conversion is different: in the first case, the goal was to integrate legacy documents

into a new XML-based authoring system, and the requirements in terms of structures were strong and fine. In the second case, the conversion into XML aims at facilitating information retrieval, and the requirements for structuring the text in a section were minimal (lines). In both cases, a specific XSL transformation was used in order to transform the XML files outputted by our conversion chain into the final customized schema.

In the first user case, the collection was composed of car repairing manuals. Each manual has about 1,000 pages. The main task was to segment the manuals into operations (the document unit). The components used were: the image extractor, the header/footer detector, the reading order component and the Toc-based structuring. The images were extracted using the component presented Section 3. One requirement was to keep the vectorial structure whenever possible, and vectorial images were converted into SVG. Ad hoc components were developed in order to associate to the image its number where occur below it. The reading order component was very important since most of the pages were two-column pages. A segmentation into paragraphs was achieved by the XY-cut-based module. Some specific structures (warnings, lists) were detected using rules. An estimation done by the customer shows that the use of this automatic system increases by 50% the conversion productivity (done manually before).

In the second user case, the problem was to structure a collection of construction contracts in divisions and sections. Even if the construction documents follow strict guidelines, the documents use various layout standards, depending on the institution. The documents are first scanned, OCR-ed and divided into divisions. Our system was used in order to segment each division into sections. The requirements for the sections were minimal: a simple structuring into lines. The components used for this user case were: the header/footer detector (essentially for the ToC pages), the reading order detector and the toc-based structuring. Each division has a table of contents with 30 to 200 entries (sections). The test set was composed of 25 documents which represent about 12,000 pages for 1600 sections. This represents 10% of a daily conversion (120,000 pages/day). The precision (at the link level) was 92% and the recall 90%. Errors are mainly due to OCR errors and mismatches between the ToC and the document body (e.g. a section in the document does not occur in the ToC).

10 Future Work and Conclusion

We have presented a system able to convert PDF documents into logically structured XML documents. The specificity of our approach relies in the exploitation of the native internal PDF objects rather than using image-based techniques on an image representation of the PDF document. We believe this is advantageous because it is computationally less intensive and potentially more efficient. Indeed exploiting the PDF native objects is helpful also in the sense that some conventional tasks such as image segmentation and labeling become either pointless or can be better done from the native objects.

Our experimentation with some real customer cases allowed us to identify some required improvements, which are now part of our future work list: other specific

components such as figure/table caption, footnotes, table, list detectors… Also, the XY-cut segmentation in paragraphs is very sensible to parameters (either a potential under-segmentation or line-segmentation). Other features like fonts, indentation have to take into account. We have also realized the importance of customizing the conversion chain in view of the customer requirements. This customization includes the conformance of the XML output with some precise schema but also requires to both parameterize appropriately each component and to determine the sequence of conversion. While the component parameterization is well supported by the GUI, the customization of the data- and control-flow within the conversion chain deserves some more support. In conclusion, we have found the proposed approach promising and are now working on the identified improvements.

References

1. Wikipedia, www.wikipedia.org
2. PDF Reference, fifth edition, Adobe® Portable Document Format
3. CambridgeDoc : www.cambridgedoc.com.
4. JPedal: www.jpedal.org
5. PDFTron, www.pdftron.com
6. K. Hadjar, M. Rigamonti, D. Lalanne, R. Ingold, Xed: a new Tool for eXtracting hidden structures from electronic Documents, DIAL'04, 2004
7. Omnipage 14, Scansoft, www.scansoft.com.
8. ABBYY FineReader, http://www.abbyy.com/
9. Adobe Acrobat Capture
10. Xpdf, http://www.foolabs.com/xpdf/
11. W Kuich, A. Salomaa. Semirings, Automata, Languages, in EATCS Monographs, on Theoretical Computer Science, Springer Verlag, 1986
12. Xiaofan Lin, Header and Footer Extraction by Page-Association, Hewlett-Packard Technical Report, www.hpl.hp.com/techreports/2002/HPL-2002-129.pdf, 2002
13. R. Cattoni, T. Coianiz, S. Messelodi, C.M. Modena: Geometric Layout Analysis Techniques for Document Image Understanding: a Review, ITC-IRST Technical Report #9703-09
14. J.-L. Meunier, Optimized XY-Cut for Determining a Page Reading Order, ICDAR, 2005
15. G. Nagy and S. Seth, Hierarchical representation of optically scanned documents, International Conference on Pattern Recognition, 1984
16. M. Aiello and A. Smeulders, Thick 2D Relations for Document Understanding. 7th Joint Conference on Information Sciences, 2003
17. T. M. Breuel, High performance document layout analysis. Symposium on Document Image Understanding, 2003
18. H. Déjean, J.-L. Meunier, Structuring documents according to their ToC, DocEng, 2005.
19. D. Dori, D. Doermann, C. Shin, R. Haralick, M. Buchman, D. Ross, I. Phillips, The representation of Document Structure. In Hansbooks on optical Character Recognition and Document Analysis, World Scientific Publishing Company, 1996.