eXEDRA: a complete Open Source Architecture for Paper Document recognition

L. Cinque, S. Levialdi, A. Malizia

Computer Science Department, University"La Sapienza" of Rome

Via Salaria 113, 00198 Rome, Italy

{cinque,levialdi,malizia}@dsi.uniroma1.it

Abstract - **The Automatic Document recognition is fundamental for office automation becoming every day a more powerful tool in those fields where information is still on paper. Document recognition follows from data acquisition, from both journals, and entire books in order to transform them in digital objects. We present a new architecture for Document recognition that follows the Open Source methodologies for documents segmentation and classification, which turns to be beneficial in terms of computation efficiency, general-purpose availability and cost.**

1. Introduction

The document analysis field is related to the semi-automatic management of paper documents. Such automation systems have been used in several fields: typically cataloguing and storing of documents, blueprints, faxing servers, character recognition software. Many different approaches have been used and standards are still lacking. Typical problems of document analysis systems are: layout segmentation, syntactic parsing, but also the selective extraction of information such as document types and semantic contents [1,2,3].
Most document processing packages are designed either for document recognition (i.e., indexing and archiving of document images) or for data acquisition (i.e., extracting data from filled forms). Document recognition is essentially the process of converting paper documents into digital images and indexing such data. Images are stored as data files (typically as TIFF files) and together with indexes are stored in a content management system [4].. Most forms oriented products available on the market today instead require the user to redesign his forms in order to achieve acceptable recognition rates. In our case we can show that the eXEDRA (Open Environment for Document Recognition Applications) framework will work both with document images and forms.

2. Modeling a document capture and analysis framework environment.

In order to model the proposed system we will briefly describe its different components:

- **A data acquisition centre could handle scanning**, where high-speed expensive equipments are used to acquire multiple format documents. Our eXEDRA environment exhibits flexibility standard definitions that allowing to change both the indexing and recognition rules even after the data acquisition phase. In fact, there are many systems on the market that use the batch process to enhance the performance and reduce costs but are wrapped around the capture process. If new requirements are needed by the customer department (for instance forms updating), the batch process must be halted and the documents scanned again with new specifications. If we have a batch of 3000 paper documents and while we have captured the first 2000 we found that something in the forms has changed (like the position of the name and surname

labels), with most of the applications we have to stop the process, change the text area specifications where the information have to be captured and then start it again, this could impact in terms of hours! Instead with our segmentation module the user can acquire batches and batches of document and have to specify the zone type, size and id thus our recognition module will detect where the zone is on the document automatically with a good impact in terms of reduction of variable costs.

- **Recognizing** is a really important in cost saving and automation. In fact using the recognition engine described in this architecture we can have automatic recognition both for images and forms. What is important here is also a standard methodology in order to introduce different recognition modules based to the market standards or research institutes results. Thus we can change our recognition modules based onto a standard definition that is the engine of this environment at run-time. In fact we can change the recognition module depending on the type of documents, what the user have to do is only to adopt the eXEDRA framework environment and develop it's own modules on top of eXEDRA specification thus writing the application quick and easily because of the existing eXEDRA framework. We can also include OCR modules to the regions obtained from the defined segmentation reducing the manual annotation in document forms.

- **Indexing** could be automatic, or manual depending onto a set of parameters that could be tuned to evaluate the segmentation and classification results. Using these parameters (which could be also set by the user) the system can decide which kind of indexing phase perform: manual or automatic. Manual indexing will require a user to data-entry information from forms, even if using automatic recognition we will have a good cost saving in terms of indexed documents per second. In fact, only those documents out of the parameters range will be presented to the user for data-entry. Moreover using both automatic and manual segmentation we could perform semi-automatic indexing, which could help in validation for manual data-entry. In order to avoid typing error, the automatic indexing is performed and then the results are presented for manual annotation, if differences appear the document has to be indexed again.

- **Storing** indexes and original document images in SQL standard database could help in fast retrieval. One of the cost saving issues here, could be found in the flexibility of the indexes definitions. In fact the engine of the architecture is a standard XML-based indexes definition schema, which could be changed even the above phases are already performed thus avoid time and production loss because of analysis or requirements changing. Moreover, using the eXEDRA framework, XML indexes description could reside only in central repository while images could be spread across multiple sites, thus saving costs in terms of: needing of central mass storage, in fact we could use a federation of different storages as we can find in every enterprise environment; and moreover users don't need to have a broadband expensive connections because images remains at the edges of the enterprise while only indexes are centralized.

- **Querying** will be performed using a standard browser with XML-parsing via HTTP protocol thus obtain a complete Open architecture. A standard Internet Browser (Microsoft Internet Explorer, Netscape Navigator, Opera, …) could be used.

3. eXEDRA: an Open Environment for Document Recognition Applications

3.1. Goals

The eXEDRA goals are a set of standard features based onto the already described modules of a Document Recognition Environment. These features are described in conjunction with the technologies used to implement those features, as we can see in **Table 1**, everything is open source and free**.**

FEATURE	GOALS	TECHNOLOGY
Segmentation and Classification	Automatic image cleaning, recognition	Standard gcc compiler, XML, OCR modules

	and classification	
Indexing	Local and remote indexing, automatic region indexing, internet-based indexing	Apache web server, PHP scripts, FTP service, MySQL and SQL based database engine
Scalability and Reliability	Expanding the system according to the requirements and security	Apache web servers, MySQL database replica, https support
Integration	Importing segmentation algorithms, choosing data format to export	DTD and XML

Table 1: eXEDRA architecture goals and technologies.

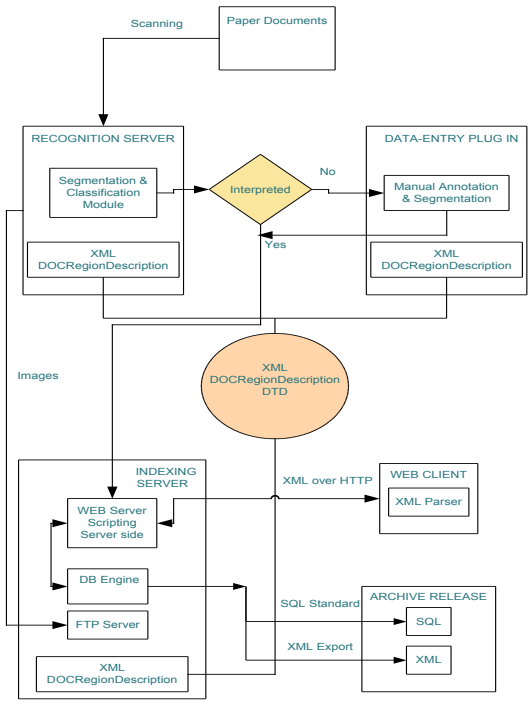

Figure 1: The eXEDRA environment general schema.

3.2. Architecture.

There are four main components in our Open document recognition environment: the Recognition Server, the Indexing Server, the Data-entry Annotation, and the Web Client (the query module). Moreover we can define also an archive release module if we want to export our indexes and images to other applications.

We can see a general draft of the architecture in **Figure 1,** where all the modules are connected to the others with relational lines.

As we can see in the general schema the core of the Open environment is the DTD document type definition, that is an open standard used in XML (eXtensible Markup Language).
One of the most time-consuming challenges for developers has been to exchange data between varieties of different systems over the Internet. Converting data to XML can greatly reduce this complexity thus creating indexes that can be read by many different types of applications. XML can also be used to store data in files or in databases. We have defined a standard description for the result of a segmentation process, in order to the let the users choose which algorithm best fits their needs. Our environment could be used both in a research and enterprise environment, in fact students and researchers that already had their segmentation algorithms could use the environment to test them; while in an enterprise environment state of art or R&D division segmentation software could be used. With this new approach only the output of the segmentation process should respect the defined format so user can write their own

algorithm with their preferred develop tools, and only the output data should be formatted using the eXEDRA DTD style. Thus the core of our system is the DTD (Document Type Definition) of the XML description file. Now what is DTD? A DTD defines the legal elements of an XML document. It defines the document structure with a list of legal elements. We use a modified version of the "RichRegionDescription" [cinque-lecca-tanimoto] called "DOCRegionDescription" including specific attributes for a document recognition environment " [5].

it performs also classification (text, image or graph region), otherwise could be set to a const fixed value, but should be very useful in terms of indexing features choosing. Instead if the algorithm provides a method for evaluating its result, for example using a range of values for the acceptance test of the segmented regions; the use of the Type attribute will help. In fact if the values are acceptable the Type value should be "Auto" and the next phase will be indexing, while for that documents where the values are out of range should be "Manual" and the next phase will be Data-entry Annotation by the user.

```
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE DOCRegionDescription SYSTEM "DOCRegionDescription.dtd">
<DOCRegionDescription Version="1.0" FileName="0002.tif" Width="640" Height="480">
<Segmentation Param1="" Param2="" Param3="" Type="Auto" Algo="Malizia">
<Region ID="1" Interpreted="text">
 <NumPixels Num="144"/>
 <MeanColor red="5" green="3" blue="4"/>
 <Size x1="452" x2="475" y1="15" y2="20"/>
 <Barycenter x="463" y="17"/>
 <Importance value="0.03779"/>
 <RegionContour PixelList="(1,1), (1,2),(1,3),(1,4)"/>
</Region>
<Region ID="2" Interpreted="graph">
 <NumPixels Num="318"/>
 <MeanColor red="188" green="21" blue="16"/>
 <Size x1="311" x2="333" y1="27" y2="62"/>
 <Barycenter x="321" y="44"/>
 <Importance value="0.08346"/>
 <RegionContour PixelList="(2,1),(2,2),(2,3),(2,4)"/>
 </Region>
<Region ID="3" Interpreted="text">
 <NumPixels Num="180"/>
 <MeanColor red="191" green="28" blue="19"/>
 <Size x1="328" x2="339" y1="27" y2="50"/>
 <Barycenter x="331" y="39"/>
 <Importance value="0.04724"/>
</Region>
</Segmentation>
</DOCRegionDescription>
```

Table 2: an image description with XML-based DOCRegionDescription

A set of standard attributes: NumPixels, Barycenter, Size, MeanColor, Texture, RegionContour, Importance and Shape, are used in order to satisfy general features of segmentation algorithms, but as we can see using a DTD as the core of our system the user can define his features and include them in the xml description file which will be used for indexing. Thus our architecture is really open even if a set of standard feature is supported, as we can see in **Table 2**. In particular, two types of attributes are important in the Open Document Recognition Environment: the Interpreted attribute of the Region tag, and the Type (Auto|Manual) of the Segmentation tag. The Interpreted attribute will be valorized from the segmentation algorithm if

Thus if a segmentation algorithm doesn't support the evaluation of parameters the Type attribute will be fixed to "Auto", while if we want to perform only manual annotation (i.e. data-entry acquisition service) useful for textual region annotation by speed typing users, the Type attribute should be fixed to "Manual". Moreover this approach could increase interoperability of different software packages, thus allowing cooperative scenarios in order to solve these kind of document recognition related problems (document analysis, image processing, segmentation, …). In fact, everyone could write it's own DocRegionDescription compliant XML specification and the application will work. In our case we have used the Pictorial Computing

Laboratory Group [dipartimento di scienze dell'informazione] results for writing the segmentation and classification engine, but writing an application compliant with the DOCRegionDescription format could be written on top of the eXEDRA environment. We could think at eXEDRA as a framework with four modules, that could be used to develop document capture and recognition applications very rapidly and easily; we provide those four modules as open source thus everyone could develop on top of this modules, with the benefits of already written communication and indexing boxes, as shown in

.

result to the Merge module, which applies pre-classification criterion in order to merge the similar regions into big regions [7,8].. We use local operators with variable threshold in order to compute this phase. Finally using global operators we have the real engine of this system in the Classification module that computes the classification procedure according to the classification logic. In fact, the "brain" of our system is this finally module, which outputs the classified document, with it's different regions, highlighted and interpreted, in XML format.

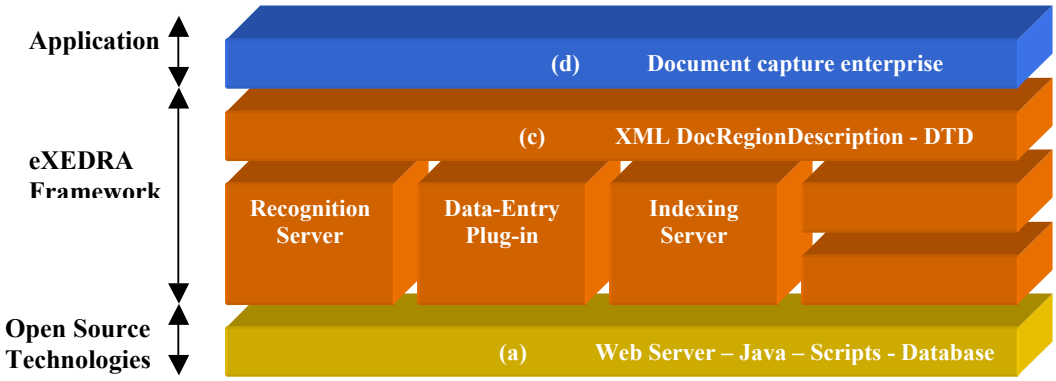

Figure 2: the eXEDRA framework.

3.3. Recognition Server.

The Recognition Server performs the automatic document recognition and indexing using the Classification and Recognition Module.

The segmentation and classification module architecture includes four main components: the preprocessor, the split module, the merge module and the classification module. The preprocessor is based onto the computation of the color histogram of the region and is used to let the other phases get the values computed by this module, in order to make the right decisions [6].. Moreover in this phase the original image of the document is loaded into main memory so obtaining better computation performances. The Split module takes input from the preprocessing phase and applies a particular quad-tree technique in order to split the document into small blocks. Then the Split module passes its

If there's at least one region "not interpreted", the Type item in the XML document will be switched to Manual and the Data-Entry Plug-in is recalled, only for those "not interpreted" regions.

As we can see, everyone could choose it's own segmentation method or algorithm, what it's required, in order to be compliant with the framework is to produce an XML output with the XML DTD format called DOCRegionDescription. Even if a starting segmentation and classification module is provided with the framework.

3.4. Data-Entry Plug-in.

The Recognition phase is based onto the Segmentation and Classification Module, which takes as input a paper document page, which is

then presented to the user if Manual indexing was decided (by the user, or by the "not interpreted" parameter) with an interface Plug-in for the Data-Entry, in our case a Java based Applet. The Data-Entry Plug-in allows users to evaluate the automatic classification performed by the system and edit the segmentation for indexing. The user can also edit the recognized regions by the classification engine and adjust their values and sizes. The output of this phase is an XML file that will be imported in the Indexing Module with an xml DB for indexing and querying. The **Figure 3** shows the Data-Entry module, with it's tools for annotating and edit the automatic recognized region sizes and properties. Obviously also images are supported as an important part of a document.

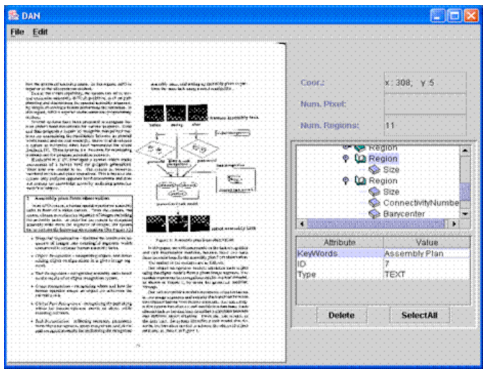

Figure 3: the Data-Entry plug-in with he annotation parameters from the XML DOCRegionDescription

3.5. Indexing Server.

The indexing server software will utilize multiple tier architecture. This section will cover the basics of such a system, as well as certain technologies used in the Indexing Server to the extent necessary to explain the architecture and the design decisions that have been met.

The Indexing Server is a web-based system that consists of several parts. The database layer stores all of the basic information kept in eXEDRA environment, such as text labels or segmentation and indexing features. The web server layer is responsible for turning the data fragments stored in the database into useful forms that are presented to the Web Client module, such as an open report, or a query result. The Web Client Module presents the forms to the user, and submits data back to web server layer.

The web server layer is the most intelligent piece of the Indexing server. This layer passes data back and forth between the Web Client Module and the database layer; generates and serves the web pages to the Web Client module; performs validation checks on user-supplied data; and manages the data that is stored in the database layer.

The Indexing server code in the web server layer is PHP. PHP is a relatively new server-side, open-source scripting language. It is platform and architecture independent, and can integrate itself with any web server software via modules for the Apache, ISAPI, and NSAPI API's, or by a generic CGI script.

The indexing server is also database independent. The database software can be set as a configuration option, and the indexing server will use the native PHP functions specific to the selected database. The indexing server will initially be realized with MySQL database software, and will also support Oracle, Microsoft SQL Server, Microsoft Access, Sybase, PostgreSQL, Visual Foxpro, Interbase, and generic ODBC.

The method the web server layer uses to communicate with the database layer will be dependent on the database software. MySQL will use a named socket on Unix systems where the database resides on the same server as the web server software. Otherwise it will use TCP/IP, via a connection established by the web server from an arbitrary port on the database server.

The database layer stores all of the information of the Indexing server. It processes SQL language queries from the web server layer, gathers together the information that was requested and returns it to the Web Client Module. The web server layer connects to the database layer using persistent connections, so avoiding overheads in establishing the communications. Key design considerations in this layer included fast retrieval and assembly of data, and a small storage footprint. This could be done through careful SQL table design to avoid storing unnecessary duplicate data.

The eXEDRA architecture allows the Indexing Server to offer a massive scalability beyond what is currently available on the market. Basic installations consisting of one server, handling both the web server and database layers, can easily handle many clients. The performance bottleneck of this setup is often the server's CPU speed.

Using a multi-processor system can help maximize the performance of this setup. SCSI interface hard drives that offload most I/O related functions from the CPU to the SCSI controller will also help, as will having sufficient RAM to cache the entire database.
Apache 1.3.19 running on a Red Hat Linux 7.1 system upgraded to the 2.4.5 kernel was able to process at peak throughput 4,602 Web requests per second, on a two-CPU server (two 700MHz Pentium III Xeon CPUs), used 30 percent dynamic Web content (which is more work to generate). Apache uses a multiple-process design instead of a multiple-thread design, which hurts its performance much more on operating systems such as Sun Microsystems Inc.'s Solaris, where it is much slower to switch between multiple processes than it is on Linux.
More advanced setups will have separate web and database servers. This can be at a 1:1 ratio, or not. Web servers can be set up as needed, using a DNS round robin setup or other load balancer to distribute the traffic among the web servers. The bottleneck on web servers in this configuration will be network I/O, as moderate, current hardware can easily saturate Fast or Gigabit Ethernet links. Many database software packages allow for database clustering, which will allow multiple database servers. Bottlenecks on database servers in this configuration will be CPU speed and disk I/O.

3.6. **Web Client.**

The client layer will consist of a W3C HTML 4.0 compatible web browser with support for JavaScript Document Object Model level 0 (DOM0) and XML parser. The W3C HTML 4.0 specification was formally released in 1997. The reference JavaScript DOM0 was implemented in Microsoft Internet Explorer 3 and Netscape Navigator 3, which have releases as early as 1997. Any system that can run a web browser that adheres to the standards listed above can serve as a client for the Indexing Server module, be it a PC, Macintosh, Unix system, thin-client, or other. The client system will parse XML code generated by the web server layer to display pages to the end user. Some user interface pieces used in the client module could be written in JavaScript, where pure HTML was not sufficient to address the requirements of the UI. The client layer will communicate with the web server layer over TCP/IP via the HTTP protocol (HTTPS, if encryption between these layers is desired) from an arbitrary port on the client system to port 80 (443 for HTTPS) to the web server layer of the Indexing Server module.

3.7. **Archive Release.**

The final stage in the production recognition process could be to release each document to a content management or workflow system. In the release process, the image files are written to permanent storage and the data is written to the target database or content management software. When dealing with forms, extracted form data can be released to a back end database or line of business application. In addition, the Archive Release module allows users to write their own custom release modules, either to modify the standard release procedure or to release documents into a proprietary back-end or non-ODBC database.

4. **Supporting the eXEDRA framework.**

Why supporting the eXEDRA framework? We will show a typical use case of an application built on the eXEDRA framework, and then we talk about configurations and calculation of ROI (Return Of Investment) supporting the presented framework.

We can see in **Figure 4**, a sequence diagram of a typical use case of an application built with our framework. After a manual or automatic indexing of a batch of documents a set of indexes compliant with DocRegionDescription format are sent to the Indexing Server (**Step 1**); using the indexing server these indexes are interpreted and then stored in tables contained in the DB engine (**Step 2**). If there is a request from the WEB Client a SQL (Standard Query Language) query is performed onto the DB engine and the indexing server format the result in order to transform them in HTML or directly send XML (**Step 3**); the HTML/XML query results are then sent via HTTP to the WEB Client for presenting the results to the user (**Step 4**).

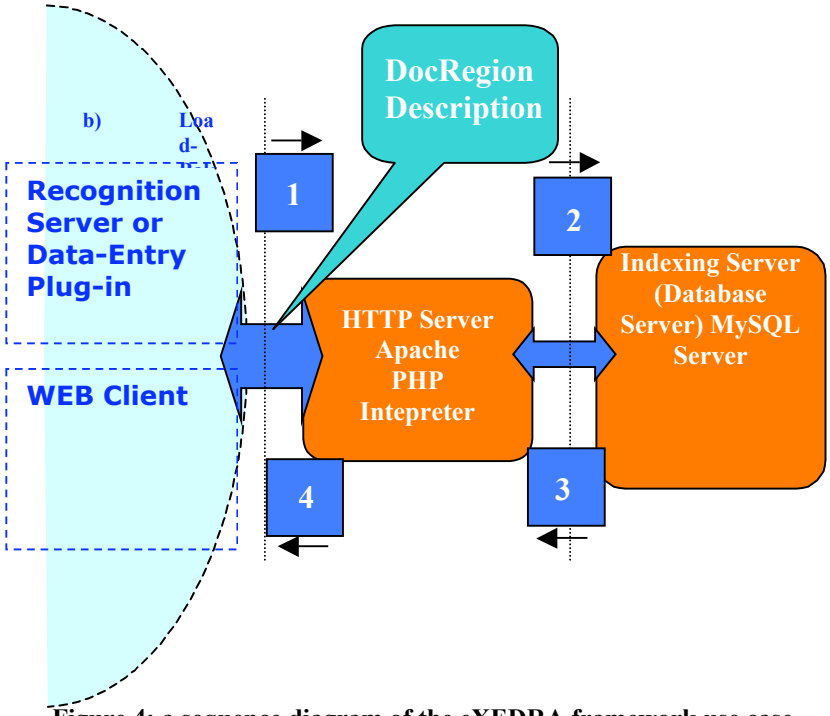

Figure 4: a sequence diagram of the eXEDRA framework use case.

A typical configuration of an Application System built onto eXEDRA framework could depend on the performance and level of geographical distribution of the customer environment but we could give some general recommendations:

1. **Recognition Server**: this module should run onto a server better dual processor and with al least 128 MB of RAM, in order to have optimal performances even with big amount of documents. In fact this module usually requires a lot of computations so is sensible to processors performances.
2. **Data-Entry Plug-in**: because it's a Java based applet, is really lightweight and could run even onto a laptop or on a thin client, thus allowing cost-savings in building a data-entry farm with finger-typing personnel.
3. **Indexing Server**: this module requires a good throughput server, so it is sensible to RAM amount (at least 256 MB) and SCSI disks having fast connection to its databases, also with Fast Ethernet connections to enhance the bandwidth with other modules.
4. **WEB Client**: it's a standard browser and could run at least on every client supporting a standard web browser.
5. **Archive Release**: this module if available could be CPU intensive because of transforming indexes in order to export them with the images to other systems.

A typical configuration will have the Recognition Server running onto a computer server on a LAN (as a file server) with many Data-Entry Plug-ins running onto the clients of the LAN. This configuration define a document capture centre that is local into one department of the organization while in another LAN, connected via Internet or Intranet there's an Indexing farm with a Load balancer and fast SCSI disks, and onto the internet or intranet worldwide there are many WEB Clients that could perform queries onto the already captured and indexed documents requesting information to the indexing farm.

5. Conclusions.

The goal is a multimedia document recognition framework that assists the user in recognizing multimedia documents also using segmentation algorithm [9] for classifying image regions, and image retrieval algorithm [10] in order build an environment for multimedia documents capture and recognition. Finally, our environment may be consider as a first step in the direction of multimedia documents recognition standard framework with region segmentation and classification, so this framework should be very suitable for automatic recognition of image database and batch acquisition of multiple multimedia documents types and formats. Moreover XTREM framework could be considered as a test bed for students and researchers that are projecting and implementing algorithms for document analysis and document capture field.

References.

[1] F. Bapst, R. Brugger, and R. Ingold. Towards an Interactive Document Recognition System. *Internal working p*aper **95-09**, IIUF-Universit´e de Fribourg, March 1995.

[2] B. Gatos, S. L. Mantzaris, K. V. Chandrios, A. Tsigris, and S. J. Perantonis. Integrated algorithms for newspaper page decomposition and article tracking. In *ICDAR'99: Fifth International Conference on Document Analysis and Recogntion*, pages **559-562**, Bangalore, India, Sept. 1999.

[3] D. Lewis. Representation and Learning in Information Retrieval. *PhD thesis*, Department of Computer Science, **University of Massachusetts**, 1992.

[4] M. Junker and R. Hoch. An Experimental Evaluation of OCR Text Representations for Learning Document Classifiers, *International Journal on Document Analysis and Recognition*, **1(2):116-122**, June 1998.

[5] L. Cinque, S. Levialdi, A. Malizia and F. De Rosa. DAN: an automatic segmentation and classification engine for paper documents, *Document Analysis Systems V*, LNCS **2423:491-502**, August 2002.

[6] G. Nagy, S. Seth, and M. Viswanathan. A prototype document image analysis system for technical journals. *Computer*, **25(7)**: 10-22, July 1992

[7] T. Ojala, M. Pietikainen. Unsupervised texture segmentation using feature distributions. *Pattern Recognition*, **Vol.32**:477-486,1999.

[8] M. Span, R. Wilson. A quad-tree approach to image segmentation which combines statistical and spatial information. *Pattern Recognition*, **Vol.18**:257-269,1985.

[9] L. Cinque, F. Lecca, S. Levialdi, S. Tanimoto – "Retrieval of images using Rich Region Descriptions". *Proceeding of the International Conference of Pattern Recognition*, Brisbane, Australia, 1998, **Volume I**, pp. 899-109.

[10] L. Cinque, S. Levialdi, A. Malizia, K.A. Olsen. "A Multidimensional Image Browser". *Journal of Visual Language and Computing*, **Vol.** 9, 1998.