Int J Digit Libr (1998) 2: 3–19

INTERNATIONAL JOURNAL ON
Digital Libraries

Original Articles

Adding machine learning and knowledge intensive techniques to a digital library service

Floriana Esposito, Donato Malerba, Giovanni Semeraro, Nicola Fanizzi, Stefano Ferilli

Dipartimento di Informatica, Università di Bari, Via Orabona 4, 70126 Bari, Italy;
E-mail: {esposito, malerba, semeraro, fanizzi, ferilli}@di.uniba.it

Abstract. This paper presents IDL, a prototypical digital library service. It integrates machine learning tools and intelligent techniques in order to make effective, efficient and economically feasible the process of capturing the information that should be stored and indexed by content in the digital library. In fact, information capture and semantic indexing are critical issues when building a digital library, since they involve complex pattern recognition problems, such as document analysis, classification and understanding. Experimental results show that learning systems can effectively and efficiently solve all these problems.

Key words: Document analysis, classification, understanding – Machine learning – Information capture – Semantic indexing

1 Introduction

The development of digital libraries requires the application of advanced methods and techniques from several disciplines. A digital library is a distributed collection of textual and/or multimedia documents, therefore one of the main problems concerns the acquisition and storing of information. The recognition of relevant parts of a semi-structured text, such as cross-references to other documents, may require a variety of methods ranging from simple text processing techniques to more complex natural language processing tools. Advanced image analysis techniques may be involved in the automated extraction of relevant features from document images or video frames. Storing multimedia documents into DBMS originally designed for business applications presents several difficulties, so that it becomes necessary to resort to the more recent database technologies. Another important problem of a digital library is the organization of data, namely the way in which documents are classified and indexed. The manual creation of an index is an unsuitable approach for the problem, because of the large volume of data. Studies in the field of text categorization can be applied in the case of textual documents, while sophisticated pattern recognition techniques may be required for images, and innovative speech understanding tools for audios. Finally, the information retrieval service provided by a digital library should also take into account the variety of users, who adopt terms from the domains about which they are most familiar. Also, the many ways in which users interact with a digital library should be captured by the interfaces, that should be adaptive with respect to the modalities in which users intend to express their needs and receive results.

A common aspect to many problems presented above is the need of knowledge in order to solve them in an efficient and effective way. Knowledge on the possible reference styles adopted in articles and books is essential to recognize cross-references between semi-structured textual documents. Knowledge on the properties of image segments enclosing textual and non-textual content is required to separate text from graphics in raster images of documents. Studies on formal languages for knowledge representation may help to solve the problem of multimedia document modeling. The automated classification of multimedia documents on the ground of extracted features is possible only if knowledge on the functions that map such features into some document classes is given. An ontology provides the system with knowledge on synonyms, hyponyms and hyperonyms of terms the user is familiar with (Goñi et al. 1997; Weinstein and Alloway 1997; Weinstein and Birmingham 1997). User models are

useful pieces of knowledge exploited by adaptive user interfaces.

In this article, we will use the term "intelligent" to characterize those digital libraries that make pervasive use of knowledge in all services provided to their different users. AI methods that acquire and manipulate knowledge play a fundamental role in providing innovative solutions to all problems of data collection, transformation, organization, representation, and retrieval that characterize a digital library (Fox 1994). In particular, machine learning methods that provide computational solutions for automatically acquiring new knowledge and for organizing existing knowledge, become crucial for the development of a truly intelligent digital library.

Building an effective and efficient digital library service is the task of a project that we started recently. This project is the natural evolution of an early project on automated document processing started in 1991 (Esposito et al. 1994). Since the beginning, it has been clear that the key issue of the project was an effective integration of tools for information compression, storage, organization, retrieval, and navigation, as well as with friendly and world-wide available standard graphical user interfaces (GUIs), and multimedia technology. One of the peculiarities of our project lies in the role that learning systems can play for information capture, semantic indexing and user modeling.

The term *information capture* denotes the task of setting information items free of the physical medium on which they are stored. When the physical medium is paper, information capturing involves the conversion of data from a paper format into a digital one. This transformation requires a solution to several problems, such as the separation of text from graphics, the classification of the document, the identification (or semantic labeling) of some relevant components of the page layout, and the transformation of portions of the document image into sequences of characters. In the literature, the process of breaking down a document image into several layout components is called document analysis, while the process of attaching semantic labels to some layout components is named document understanding (Tang et al. 1994). Furthermore, the term document classification has been introduced to identify the process of attaching a semantic label (a class name) to the whole document on the ground of its layout alone (Esposito et al. 1994). These three processes are preliminary to the application of an Optical Character Recognizer (OCR) to specific portions of the document bitmap with the aim of extracting the text. In this way, we aim at the high-level understanding of the semantic content of the document, which is the conditio sine qua non to index the information in the library according to its content (semantic indexing).

This paper presents IDL (Intelligent Digital Library), a prototypical intelligent digital library service that uses several learning systems to support both the information capture and human-computer interaction. In the next section, the role of AI methods in the problem of converting data from a paper format into a digital one is addressed. In particular, we describe the functional architecture of WISDOM, the interface used by IDL to perform the four tasks of document analysis, classification, understanding, and text extraction. The application of the learning systems integrated in IDL to the problems of document classification, document understanding and user modeling are presented in Sect. 3. Learning systems are just some of the application enablers of IDL: a complete description of the architecture of IDL, of the object model of the repository, and of the (meta-)query language defined for this model are given in Sect. 4. The different roles of the users of the digital library, namely administrator, librarian and end-user are explained in Sect. 5, while the two levels of adaptivity of end user interfaces are illustrated in Sect. 6. Section 7 reports the conclusions and the plan of the future work on IDL.

2 Intelligent techniques for paper document processing

WISDOM (Windows Interface System for DOcument Management) is the interface used by IDL as the front-end to transform printed information into a symbolic representation (Malerba et al. 1997b). This transformation process is performed in four distinct steps: document analysis, document classification, document understanding and, finally, text recognition with an OCR. The distinguishing feature of WISDOM is the use of a knowledge base that is automatically built using several machine learning tools and techniques. Each authenticated user of the interface has his/her own knowledge base. Rules in the knowledge base are applied in the first three processing steps, while rules used in the text recognition phase are embedded in the commercial OCR linked to the interface.

WISDOM can manage multi-page documents, each of which is a sequence of pages. The definition of the right sequence is the responsibility of the user, since the scanner is able to scan a single page at time. Pages of multi-page documents are processed independently of each other in all steps, though results concerning distinct pages of the same document are finally grouped in a single file. For this reason, the following description of the document processing flow will mostly concern single pages, with some occasional reference to problems related to multi-page documents.

2.1 Pre-processing and segmentation

The processing flow of WISDOM is shown in Fig. 1. Initially, each document page is scanned with a resolution of 300 dpi and thresholded into a binary image. The bitmap of an A4-sized page takes 1.092 MB and is stored in TIFF format. In order to convert the pixel representation of

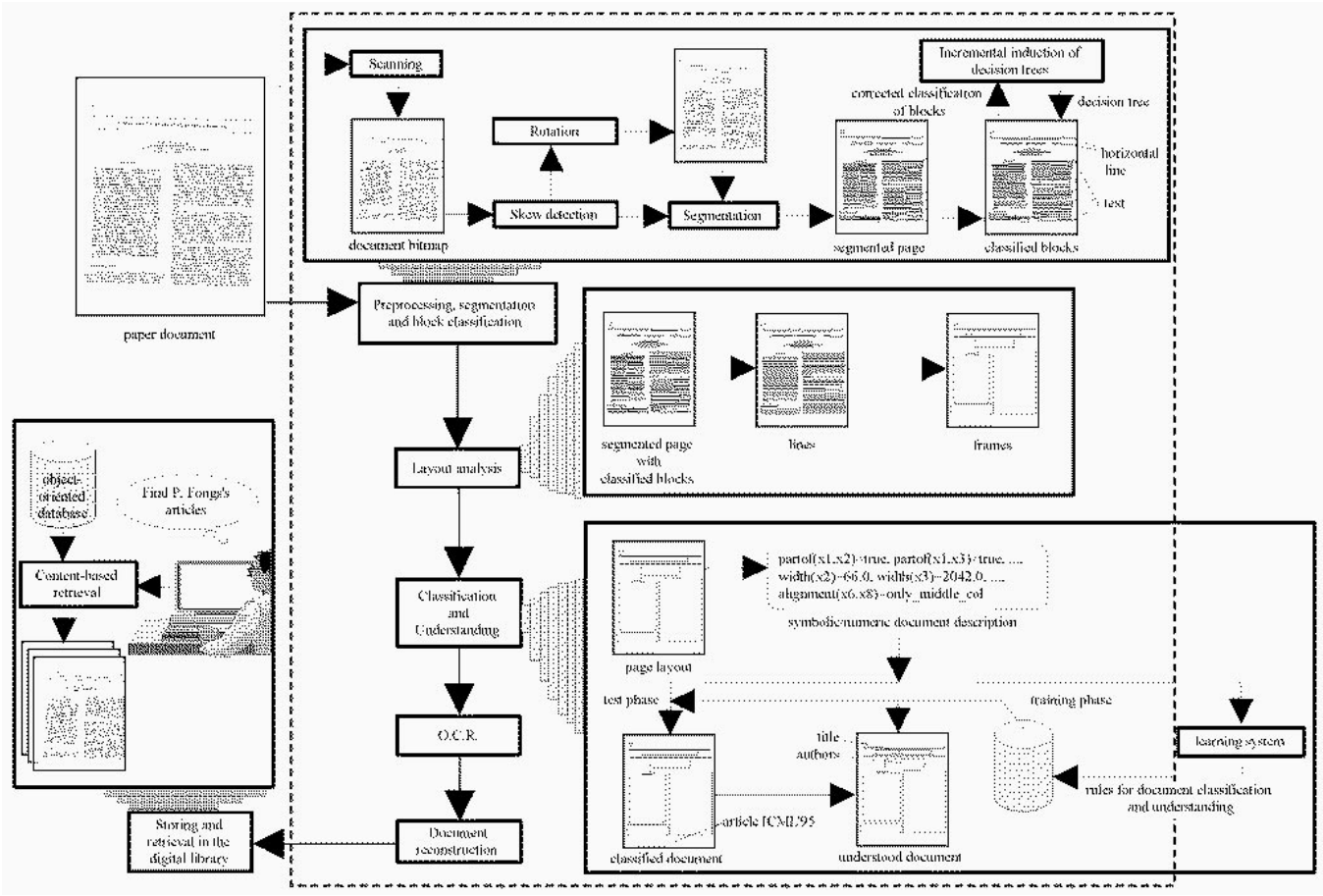

Fig. 1. WISDOM processing flow

the page into a structured set of symbolic entities, which are appropriate for computerized information processing, the system must segment the page as the first step. However, the result of the segmentation algorithm used in WISDOM depends on both the skew angle (i.e., the orientation angle of text lines in the page), and the choice of some critical parameters. To make the system less vulnerable to non-zero skew angles and/or arbitrary parameter definitions, some pre-processing algorithms have been integrated.

In particular, the skew angle is detected by means of a study of the horizontal projection profile of the page image. This profile is a histogram that represents the total number of black pixels for each row of the bitmap. The histogram has sharply rising peaks with width equal to the character height when text lines span horizontally, while it shows smooth slopes when the raster image has a large skew angle. Thus the skew angle can be estimated as the angle that maximizes the mean square deviation of the histogram. The peak-finding algorithm used in WISDOM is described in (Ciardiello et al. 1988). The estimated skew angle is finally suggested to the user when he/she invokes the 'Rotate' function from the menu.

The study of the horizontal profile of the document allows WISDOM to estimate the complexity of the document as well. This parameter is computed as the ratio of the mean distance between peaks and the peak width, and it is greater than (lower than) 1.0 for simple (complex) documents. The complexity factor affects the smoothing parameters used in the segmentation phase. The estimated complexity is suggested to the user when the 'Analyze layout' function is invoked.

The page is segmented by means of a fast top-down technique called Run Length Smoothing Algorithm (RLSA) (Wong et al. 1982). This algorithm consists of four steps: 1) horizontal smoothing; 2) vertical smoothing; 3) logical AND of the two smoothed images; 4) final horizontal smoothing. Parameters used in the smoothing operations depend on the complexity of the document. In order to speed up the segmentation process, the RLSA does not operate on the original bitmap, but on a reduced document image with a resolution of only 75 dpi (70 KB for an A4-sized document). The result of the segmentation process is a list of rectangular areas, called blocks, corresponding to printed areas in the page image. Each block is easily described by a pair of coordinates, namely top left-hand corner and bottom right-hand corner. Note that the assumption that printed areas are rectangular encompasses the assumption that all text lines have no skew. This explains

the need of evaluating the page skew and rotating the image.

2.2 Machine learning for block classification

Page segmentation identifies blocks that may contain text or graphic information. It is important to label these blocks according to the type of content. In this way the separation of text from graphic data is complete, so that subsequent processing stages may operate exclusively on the appropriate type of information (e.g., an OCR will be applied only to textual components).

This separation problem can be reformulated as a classification (or discrimination) problem, where the classes are *text block*, *horizontal line*, *vertical line*, *picture* (i.e., halftone images), and *graphics* (e.g., line drawings). Traditionally, linear discriminant analysis techniques have been used to classify blocks (Wong et al. 1982; Wang and Srihari, 1989; Shih and Chen, 1996); the only exception is the rule-based classification method by Fisher et al. (1990). In WISDOM, a decision tree classifier performs this task, but, unlike the approach of Fisher et al., the classifier is not hand-coded. The decision tree is induced from a set of training examples (blocks) of the five classes. Only ten numerical features are used to describe each block, namely: 1) height; 2) length; 3) area; 4) eccentricity; 5) total number of black pixels in the reduced bitmap; 6) total number of black pixels in the segmented block; 7) number of white-black transitions in the reduced bitmap; 8) percentage of black pixels in the reduced bitmap; 9) percentage of black pixels in the segmented block; 10) mean horizontal length of the black runs in the reduced bitmap. The choice of a "tree-based" method instead of the more common generalized linear models is due to its inherent flexibility, since decision trees can easily handle complicated interactions among features and give results that are simple to interpret.

In a previous study, a decision tree was induced using a set of 5473 examples of pre-classified blocks obtained from 53 documents of various kinds. The system used for this learning task was an extension of C4.5 (Quinlan 1993) that implemented several techniques for decision tree pruning. A study of different decision tree pruning methods in this domain of block classification has shown that pruning is generally beneficial, since it reduces the size of the decision tree and increases its predictive accuracy (Esposito et al. 1997a). The best result we observed featured an average accuracy above 97%, so we decided to embed the decision tree classification procedure into an early version of WISDOM.

The main limit of this solution is that the decision tree learner operates offline: it is not possible to revise the decision tree when some blocks are misclassified, unless a new tree is generated from scratch using an extended training set. Furthermore, some blocks can be considered text for some users and graphics for others, as in the case of a logo. To give the user the possibility of training the system online, systems for the incremental induction of decision trees have been taken into consideration. Such systems revise the current decision tree, if necessary, in response to each newly observed training instance. Many of them have been presented in the literature, namely ID4 (Schlimmer and Fisher 1986), ID5R (Utgoff 1989), and ITI (Utgoff 1994). The system that we have currently embedded in WISDOM is ITI, which is the only incremental system able to handle numerical attributes such as those used to describe the blocks.

In the *normal* operation mode, ITI first updates the frequency counts associated to each node of the tree as soon as a new instance is received. Then it restructures the decision tree according to the updated frequency counts. In this operation mode, ITI builds trees with almost the same size and the same predictive accuracy as those produced by C4.5. In fact, in a 10-fold cross-validation experiment on the above mentioned set of 5473 examples of pre-classified blocks, we observed the following results:

	Average no. of nodes	Predictive accuracy
C4.5	89.8	96.80
ITI 2.5	92.4	96.88

In this experiment, both systems were allowed to prune the induced trees using their default pruning methods (error-based pruning for C4.5 and minimum description length pruning for ITI). We can conclude that the choice of ITI as the decision tree induction system does not affect the accuracy of WISDOM.

The normal operation mode guarantees building the same decision tree independently of the order in which examples are presented. Despite this interesting property, it has been observed that ITI creates huge files (more than 10 MB) when trained on the set of 5473 instances used in the feasibility study. The reason for this space inefficiency is due to the need of storing frequency counts of all attributes of training examples in each node of the induced tree. This inefficiency can be contained (about 1.5 MB) when the system operates in the error-correction mode, since frequency counts are updated only in case of misclassicafion. In this way, however, the independence of the induced tree from the order of presentation of training instances is no longer guaranteed. Luckily, we noticed that a single user can obtain satisfying results with a lower number of training instances, since printed documents managed in a specific application often have a similar layout. For instance, on a set of 112 documents used in another experiment, we obtained high accuracy with decision trees that take less than 80 KB. Therefore, ITI has been integrated in a new release of WISDOM, and two new functions have been added to the interface: the interactive correction of the results of the block classification, and the revision of the block classifier.

2.3 Knowledge-based layout analysis

A segmented page is certainly easier to manage than the original bitmap, since the number of blocks is generally less than one hundred. Switching from the raster image space to the corner space turns out to be of great computational advantage. However, this new page representation is still too detailed. Generally, we do not need so much information for the subsequent phases of document classification and understanding. Layout analysis is the perceptual organization process that aims to detect structures among blocks. The result is a hierarchy of abstract representations of the document image, called the geometric (or layout) structure of the document. The leaves of the layout tree (lowest level of the abstraction hierarchy) are the blocks, while the root represents the whole document. In multi-page documents, the root represents a set of pages, where a page is a rectangular area that encloses printed information of a single bitmap. A page may group several layout components, called frames, which are still rectangular areas of interest in the image of a document page. An ideal layout analysis process should produce a set of frames, each of which can be associated with a distinct semantic label, such as title and author of a scientific paper. In practice, however, a suboptimal layout structure in which it is still possible to distinguish the logical meaning of distinct frames should be considered a good output of the layout analyzer.

Many approaches have been proposed for the extraction of the layout structure from the digital image. In WISDOM we adopted a knowledge-based approach that exploits generic knowledge and rules on typesetting conventions in order to group basic blocks together into frames (Malerba et al. 1995). Such knowledge is independent of the particular class of processed documents and turns out to be appropriate for a range of problems.

More precisely, the layout analysis is performed in two distinct steps (Esposito et al. 1995):

1) A global analysis of the document image in order to determine possible areas containing paragraphs, sections, columns, figures and tables. This step is based on an iterative process in which the vertical and horizontal histograms of text blocks are alternatively analyzed in order to detect columns and sections/paragraphs respectively.

2) A local analysis of the document aiming at grouping together blocks that possibly fall within the same area. Perceptual criteria considered in this step are:

- *Proximity:* Adjacent components belonging to the same column/area are equally spaced.
- *Continuity:* Overlapping components.
- *Similarity:* Components of the same type, with an almost equal height.

Pairs of layout components that satisfy some of these criteria may be grouped together. In the grouping process the type of information of composing blocks is kept whenever possible. Each layout component is associated with one of the following type: text, horizontal line, vertical line, picture, graphics, and mixed. More precisely, when the constituent blocks of a logical component are homogeneous, the same type is inherited by the logical component, otherwise the associated type is set to mixed. In Fig. 2, the output of the document analysis process performed by WISDOM is shown. By clicking on the radio buttons 'text', 'horizontal line', 'vertical line', 'image', 'graphics', and 'mixed' it is possible to only display a particular type of logical component, while by clicking on the radio buttons 'Frame 2', 'Frame 1', 'Set of Lines', 'Lines', and 'Basic blocks' it is possible to choose a level of the layout hierarchy to be displayed. The result of the layout analysis process is a file describing the hierarchy of layout components, made up of blocks (at the bottom), lines, sets of lines, first frames, and second frames (at the top).

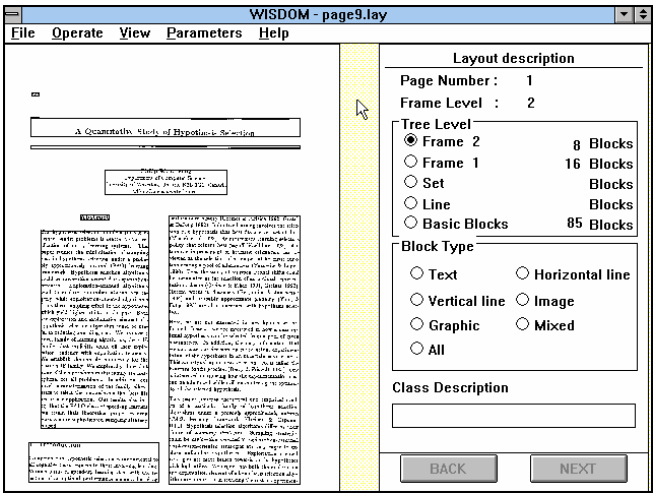

Fig. 2. Document analysis

2.4 The rule-based system for document classification and understanding

After having detected the layout structure, the logical components of the document, such as title, authors, sections of a paper, can be identified. The logical components can be arranged in another hierarchical structure, which is called *logical structure*. The logical structure is the result of repeatedly dividing the content of a document into increasingly smaller parts, on the basis of the human-perceptible meaning of the content. The leaves of the logical structure are the basic logical components, such as authors and title. The heading of an article encompasses the title and the author and is therefore an example of a composite logical component. Composite logical components are internal nodes of the logical structure. The root of the logical structure is the document class, such as an article published in the ICML Proceedings. Currently, WISDOM supports two-level logical structures, where the document class is the only composite logical component.

The problem of finding the logical structure of a document can be cast as the problem of associating some layout components with a corresponding logical component. In WISDOM this mapping is limited to the association of a page with a document class (document classification) and the association of second frames with basic logical components (document understanding). The computational strategy adopted for understanding a document consists of a hierarchical model fitting, which limits the range of labeling possibilities. More precisely, the document is first matched against models of classes of documents and then against models of the logical components of interest for that class. Since models are rules expressed in a first-order logic language, the operation of model fitting becomes a classical matching test between a logic formula that describes a model and another logic formula that represents the document layout. Indeed, WISDOM describes the frame 2 level of the layout hierarchy by means of the following:

- *attributes,* such as the height of a frame (numeric), width of a frame (numeric), type of a frame (text, horizontal line, and so on), coordinates of the centroid of a frame (numeric).
- *binary relations,* such as part-of, on-top, to-right (boolean), and relative alignment (only by left column, only by right column, by both columns, by middle column, only by upper row, only by lower row, by both rows, and by middle row).

In Fig. 3, a partial description of the page layout of the document in Fig. 2 is reported. The description is a logical conjunction of literals of the form:

$$f(t_1, \cdots, t_n) = \text{Value}$$

where f is an n-ary function symbol, called *descriptor*, t_is are constant terms, and Value is one of the possible values of f's domain. The choice of a first-order logic language answers to the requirement of flexibility and generality.

Both the rules used for the classification and understanding process are automatically learned from a set of training documents for which the user has already provided the correct class and frame labels. By looking at each class or frame label as a distinct concept to be learned, it is possible to apply conceptual inductive learning methods whose final products are rules expressed in high-level, human-oriented terms and forms. The conceptual learning systems used to generate such rules for WISDOM are described in the next section.

```
PART_OF(X1,X2)=true, PART_OF(X1,X3)=true, ...,
PART_OF(X1,X9)=true, WIDTH(X2)=66.0,
WIDTH(X3)=2042.0, ..., WIDTH(X9)=986.0,
HEIGTH(X2)=34.0, HEIGTH(X3)=150.0, ...,
HEIGTH(X9)=306.0, TYPE(X2)=text, TYPE(X3)=mixture, ...,
TYPE(X9)=text, X_POS_CENTRE(X2)=246.0,
X_POS_CENTRE(X3)=1230.0, ...,
X_POS_CENTRE(X9)=686.0,
Y_POS_CENTRE(X2)=386.0,
Y_POS_CENTRE(X3)=632.0, ...,
Y_POS_CENTRE(X9)=3162.0,TORIGHT(X9,X7)=true,
ALIGNMENT(X2,X3)=only_left_col,
ALIGNMENT(X3,X4)=both_columns, ...,
ALIGNMENT(X6,X8)=only_middle_col
```

Fig. 3. First-order description of a document

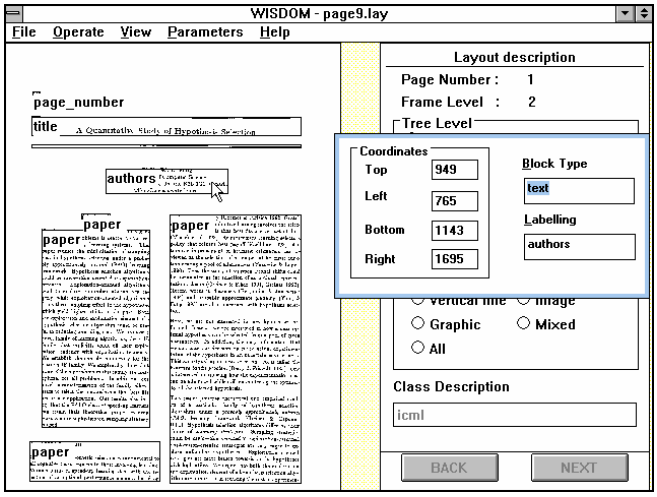

Fig. 4. Classification and understanding

Once the logical components have been detected (see Fig. 4), the system allows the user to set up the text extraction process by selecting the logical components to which the OCR has to be applied. Selective application of an OCR to variable areas of the raster image makes the interface an appropriate front-end for digital libraries. The result of document processing is a text file that contains all the relevant information on the original document image (namely, the heading of the original TIFF file), on the layout structure, on the class and frame labels, as well as on the text read in some frames by the OCR. This file with extension ".lay" can now be used to feed the digital library.

3 The learning server

The learning systems that perform the task of automatically acquiring the classification patterns (document classification) and the theory for the "semantic" indexing of documents (document understanding) constitute some of the application enablers in the system architecture of IDL, a prototypical intelligent digital library service (Semeraro et al. 1997a, 1997b). Machine learning methods could also be used to improve the user interaction allowing the design of adaptive interfaces. Indeed it is possible to induce the user interaction model by capturing the appropriate raw data for representing and classifying the individual user's observed behavior. The inferences that can be drawn from the analysis of the interactions are useful to determine the changes (adaption) which the Digital

Library system can accomplish to respond to the information needs of a specific "recognized" user. In this section the conceptual learning systems for document classification and understanding are described as well as the user modeling activity aiming at automatically acquiring the classification patterns of the typical users of Digital Library services.

3.1 Machine learning for document classification and understanding

As mentioned above, the document classification patterns and the semantic labels are automatically induced from the first-order descriptions of the layout of some training documents: each training document is associated with a class and a set of labeled layout components. Therefore, two learning problems can be defined: first, induce models of classes of documents, and then find models of the logical structures of each class.

Initially, a supervised inductive learning system that implements a hybrid approach was applied to the document classification task (Esposito et al. 1990). In this experiment, the layout of each document was described by numeric and symbolic features. A parametric method for linear classification used the numeric features, while a conceptual method induced some models from the symbolic features alone. The combination of the predictions made by the two methods provided the best results in terms of simplicity and predictive accuracy of the learned models. This hybrid approach operates in a batch way: all models are learned from scratch each time the learning process is activated and the batch strategy is the best way to build the "initial knowledge base" that represents the classification patterns resulting from a training phase based on a statistically significant number of examples. More recent experiments showed promising results also for an incremental approach in which models are progressively specialized and generalized as new incoming documents are misclassified or not classified at all (*theory revision*) (Semeraro et al. 1995). This is a very important issue, since the dynamic nature of digital library applications requires a progressive adjustment of the induced classification patterns instead of learning the whole theory from scratch every time a new document is added to the repository. Thus the best strategy is to learn an initial theory in a batch way and then refine it incrementally any time new documents become available. When the number of refinements becomes high, the theory will be learned again from scratch from the whole set of available documents.

These results and motivations led us to closely examine the effectiveness of the incremental methodologies on the problem of classifying scientific papers. In this setting, the layout of the first page of each paper is represented by a first-order logic description such as that in Fig. 3. We considered a database of 92 scientific papers, belonging to three different classes, namely ISMIS (Proceedings of the International Symposium on Methodologies for Intelligent Systems), PAMI (IEEE Transactions on Pattern Analysis and Machine Intelligence), and ICML (Proceedings of the International Conference on Machine Learning). Each paper is a positive example for the class it belongs to and, at the same time, is a negative example for all the other classes. For each class, the learning process has been performed both in a batch and in an incremental way. In the latter case, the generalization and specialization tasks have been run separately. The experiments have been replicated 33 times, by randomly splitting the database of 92 papers into two subsets, namely a learning set and a test set. In turn, the learning set has been subdivided into training and tuning sets. The learning set has been exploited in two distinct ways, according to the mode – batch or incremental – adopted for the learning process. For the batch mode, this set has been entirely given to INDUBI/CSL (Malerba et al. 1997a), an empirical learning system adopting the VL_{21} representation language (Michalski 1980), with both examples and rules expressed as first-order normal clauses. It implements a separate-and-conquer search strategy at a higher level while at the low level a beam search is performed. More precisely, INDUBI/CSL starts with a seed-example of the target concept and generates a set of the best generalizations that are consistent with respect to all of the negative examples. Then the best of these generalizations is chosen according to some heuristics (Michalski 1980) taking into account several criteria. The positive examples that are covered by such a generalization are removed from their set and the learning algorithm is restarted on this smaller set to generate other rules for the same concept. At the low level, each generalization is built by adding to the most general clause a subset of the seed-example literals, with constants previously turned into variables, provided that the rule remains consistent with the negative examples.

For the incremental mode, only the training set has been used in order to produce the initial classification theory, while the tuning set has been exploited to incrementally correct omission and commission errors made by the theory, if any, through the incremental learning system, called INCR/CSL (Esposito et al. 1997b). This systems adopts the same representation language used by INDUBI/CSL while the learning algorithm implemented is incremental. The aim is to compute the target classification rules by a progressive refinement of the starting rules through operators performing generalization or specialization, when the new/next (positive or negative, respectively) training example is not correctly classified by the current version of the rules. Unlike INDUBI/CSL, such a learning algorithm can exploit a previously generated version of the rules, reducing in this way the learning effort needed.

The tuning set is made up of only positive (negative) examples in every run concerning the generalization (spe-

Table 1. Results on the problem of document classification

		average time	min. time	max. time	average PA.	min. PA.	max. PA.	average difference time	average difference PA.
ISMIS	batch	137.424	68	208	92	81	100	–	–
	incr-gen	67.879	24	129	90	74	100	69.545	2
	incr-spec	56.788	36	108	97.061	85	100	80.636	−5.061
PAMI	batch	94.364	65	135	94	81	100	–	–
	incr-gen	58.03	26	209	94.212	78	100	35.334	−0.212
	incr-spec	67.363	45	127	91.394	81	100	27	2.606
ICML	batch	96.576	57	133	93.636	81	100	–	–
	incr-gen	81.109	32	216	90.697	81	100	13.666	2.939
	incr-spec	94.182	44	152	94.636	81	100	1.394	−1

cialization). Lastly, the test set has been exploited to evaluate the predictive accuracy of the learned theories on unclassified documents.

Table 1 shows the results obtained by comparing the theories learned in batch mode with those learned incrementally along two dimensions, namely their predictive accuracy on the test set and the computational time taken by the learning system to produce the theories. The goal of this comparison is to see if incrementally adjusting the theory on the grounds of new evidence generates polymorphic classification patterns, decreases the predictive accuracy and significantly affects the computation learning time. Specifically, the batch time is relative to the training set for the batch mode, while the incremental time is computed as the sum of the computational time concerning the training set for the incremental mode plus the time concerning the tuning set. Values concerning the predictive accuracy are percentages, while those concerning the time are expressed in seconds. All the values refer to the averages on the 33 runs.

Table 2 illustrates the results of the t-test, a statistical method exploited to evaluate the significance of the observed differences as to predictive accuracy and computational time for each class. This test has been performed as a two-sided test at a 0.01 level of significance. Each entry in the table contains the t-value and the corresponding significance value.

Table 2. Statistical results on the problem of document classification

		time		PA.	
		t value	signif. %	t value	signif. %
ISMIS	incr-gen	10.833	0.0001	2.201	0.351
	incr-spec	9.954	0.0001	−4.289	0.0002
PAMI	incr-gen	6.379	0.0001	−0.229	0.8201
	incr-spec	8.409	0.0001	3.139	0.0036
ICML	incr-gen	1.802	0.0809	3.39	0.0019
	incr-spec	0.490	0.6276	−1.22	0.2315

It is possible to note that, on the grounds of the results obtained by the t-test, the batch-learned theories and the incrementally-learned ones are comparable as to predictive accuracy: in some cases the difference is statistically significant in favor of the batch system, in other cases it is in favor of the incremental one, and in others there is no statistically significant difference. On the contrary, there is an improvement when learning the theory in incremental mode with respect to learning it in batch mode, as regards the computational time.

Table 3. Results on the problem of document understanding

		average time	min. time	max. time	average PA.	min. PA.	max. PA.	average difference time	average difference PA.
title	batch	823.576	228	1483	91.273	84	97	–	–
	incr	334.788	65	1578	86.394	76	95	488.79	4.879
authors	batch	606.758	183	1238	92.909	84	98	–	–
	incr	388.03	88	1768	86.788	74	95	218.73	6.121
abstract	batch	434.576	90	876	95.212	85	100	–	–
	incr	172.788	23	838	91.485	84	98	271.79	3.727
paper	batch	753.515	168	1802	93.394	82	100	–	–
	incr	327.03	48	1786	90.061	80	98	426.49	3.333

Table 4. Statistical results on the problem of document understanding

	time		PA.	
	t value	signif. %	t value	signif. %
title	6.532	0.0001	5.389	0.0001
authors	3.585	0.0011	6.89	0.0001
abstract	6.7	0.0001	3.983	0.0004
paper	5.849	0.0001	3.138	0.0036

As to the document understanding task, a different approach has been tested. In this case, indeed, we have to learn models of the logical components which refer to a part of the document rather than to the whole document. Again, we can compare the theories learned in batch mode to those learned incrementally (Table 3), and we perform the t-test on those results in order to evaluate the significance of the observed differences (Table 4). In this case there is a statistically significant difference in favor of the batch-learned theories as regards predictive accuracy, but the great gain in computational time in favor of the incremental system suits our application requirements.

3.2 Machine learning for user modeling

A fundamental problem to cope with when developing a system used by several people is to be able to recognize user profiles in order to adapt the interface to user needs, with the aim of improving the overall *usability* of the system.

Machine learning techniques can be used to infer user models with the aim of building an intelligent agent that collects the behavior of the typical users of a digital library service, provides each user with an interface that can speed up the process of understanding the organization and the content of the chosen digital library, and properly assist him/her during all the steps necessary to retrieve the desired information. Specifically, the main function of the Learning Server is to automatically assign each user of a digital library to one of some pre-defined classes, on the grounds of information drawn from real interaction sessions with the system. In the literature of human-computer interaction, this activity is known as *interaction modeling* (Banyon and Murray 1993).

The reasons why a user consults a digital library can be the most disparate ones, from real needs for bibliographic search, to interpreting data, following cross references, checking the content relations and information overlapping, personalized exploitation etc. After all, each user has his/her own profile, thus when using the system, he/she will behave differently from any other user. Obviously it is impossible, even for an intelligent system, to adapt its behavior to each single user. Nevertheless, it is desirable to use intelligent techniques in order to understand which kind of remote user it is interacting with: in this way, it becomes possible to design more effective help facilities (through contextual helps, different interaction modalities, personalized dialogues, etc.). Our approach takes advantage of machine learning methods and techniques, as pointed out by recent work in literature (Moustakis and Hermann 1997; Pazzani and Billsus 1997). Specifically, interaction modeling can be cast as a supervised learning problem by considering user interactions with the digital library as training examples for a learning system, whose goal is to induce a theory for classifying the users.

The first step requires the definition of the classes of users that seem to be meaningful for the system, and the identification of the features that properly describe them, with the aim of better characterizing each class of users and discriminating it from all the other classes.

In a first experiment, among all the possible generic users of a digital library, we choose only the class of the end users, namely *Novice*, *Expert* and *Teacher*, based upon their different skills and motivations. This identifies three distinct groups of users, characterized by a different (growing) degree of familiarity with the specific library service and, more generally, with the domain of libraries and with online access to document collections.

In order to choose a suitable language for representing user interactions, all the characteristics that could be necessary and useful to understand the type of user were investigated; the final selected features should be detected from the raw data stored in a file associated to each user. Basic relevant characteristics are those concerning the way in which users exploit the capabilities of the digital library search engine, such as date and beginning time of session, the specific class of documents chosen, search index(es) used, the criterion selected to sort the results of the search, the number of documents obtained as result of the search, and the types of errors performed when interacting with the system. The formal description of these characteristics with the user class label consitute the training data for a supervised learning system.

Specifically, the learning system C4.5/C4.5RULES (Quinlan 1993) is used in order to induce a decision tree and a set of rules allowing the system to autonomously classify the users interacting with digital library services.

Whenever any client connects to the digital library as a user of the system, the corresponding interaction information (*logs*) is exploited to generate a new example that the learning server will classify on the ground of the inferred rules. The way in which rules are consulted by the learning server (and the existence of the classification rules themselves) is completely transparent to the user of the system.

C4.5/C4.5RULES has been customized in order to work in a batch way to infer the classification theory from the log files selected to train the system. The scheme according to which the classification rules are induced (*training phase*) and afterward used to classify a user (*operational phase*) is given in Fig. 5. Furthermore, we are

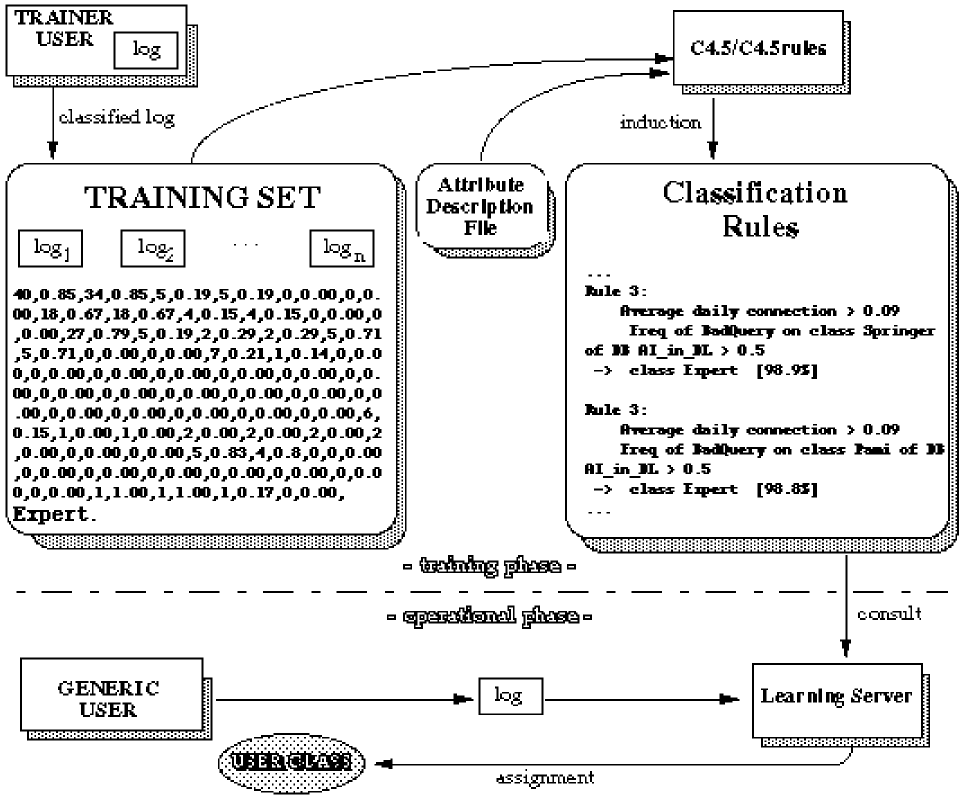

Fig. 5. Induction/classification through the learning server

currently investigating the possibility of using incremental learning systems (Esposito et al. 1997b) that avoid the drawback of starting the learning process from scratch each time new log examples become available.

4 The architecture of IDL: an insight

A digital library service is a set of modules that can be classified as either resource managers or application enablers. A *resource manager* is a program that represents the only access path to the data contained in a protected resource and is accessible to multiple, concurrent clients. Intuitively, a protected resource is a data collection. An *application enabler* is a software that allows a class of users to make application programming easy and quick (or to avoid it completely). The logical architecture of IDL is shown in Fig. 6. The Repository is a protected resource containing the actual collection of data that constitutes the digital library. Usually, it consists of highly structured items. Here, we use the word database rather than information collection because in IDL it is a commercial object-oriented DBMS, namely ObjectStore by Object Design, Inc. Thus, the Repository is actually a set of objects. More precisely, these objects constitute an instance of an ObjectStore conceptual scheme that we designed purposely for IDL. The document object model is the conceptual schema according to which documents are

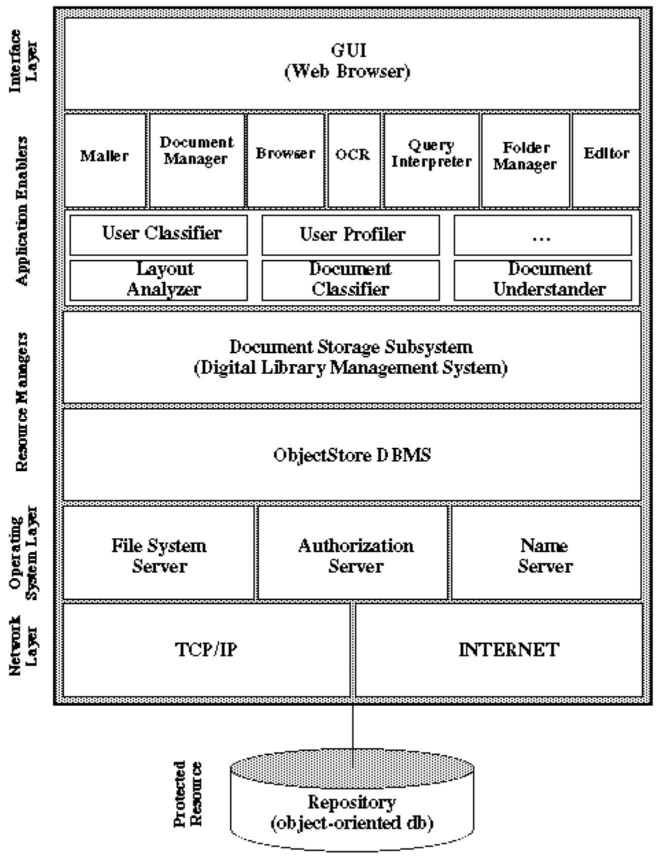

Fig. 6. IDL architecture

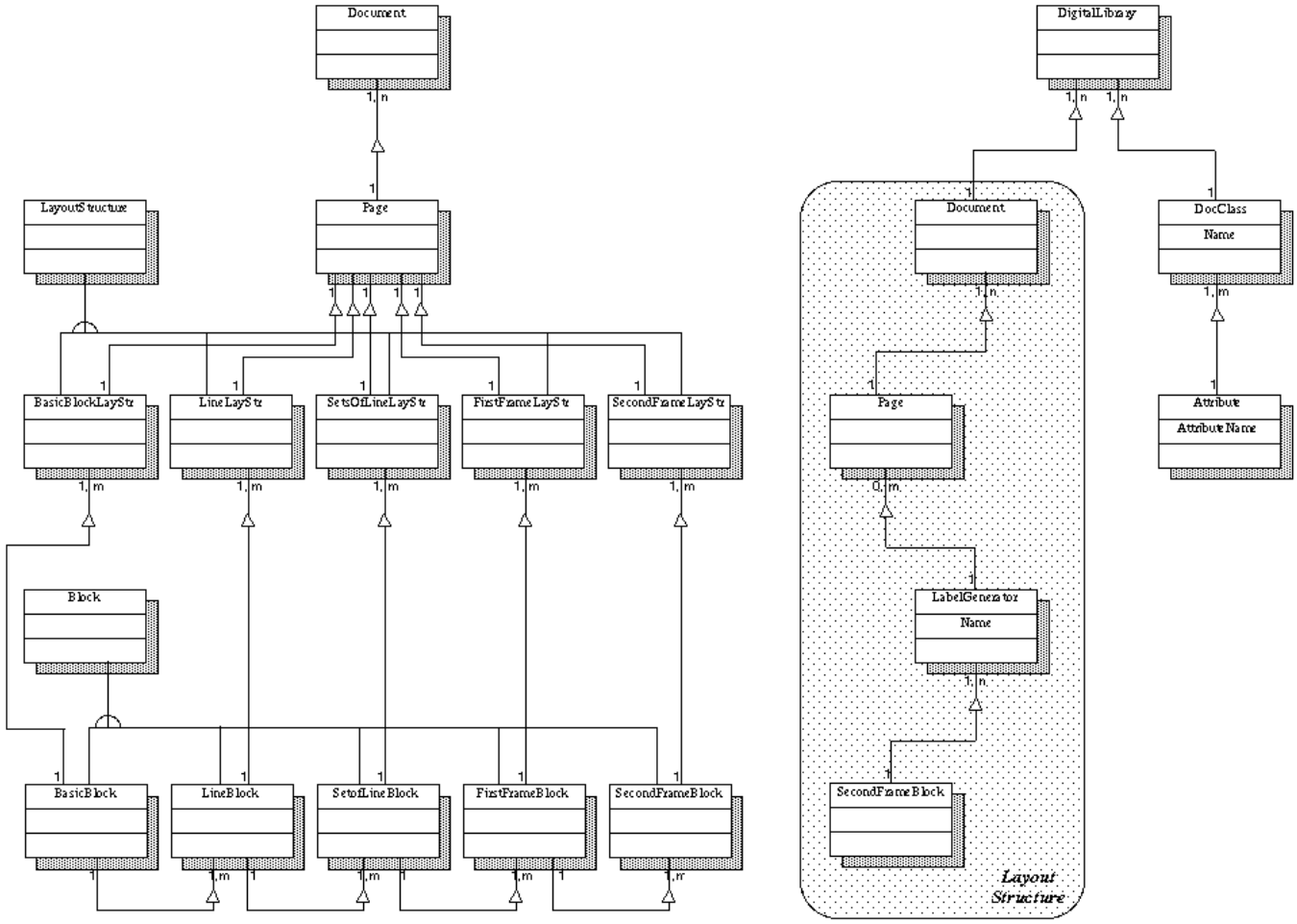

Fig. 7. Layout Structure and Logical Structure Object model

stored and (internally) represented in both the layout and the logical structure.

The portion of the object model that allows us to store (and retrieve) the objects related to the layout structure of a document is given in Fig. 7a. For our purposes, we defined a hierarchy of five distinct classes corresponding to the levels of components in the layout tree (other than *Page*): *BasicBlock*, *LineBlock*, *SetOfLineBlock*, *FirstFrameBlock*, and *SecondFrameBlock*. All the objects at any level share the same structure, therefore the five classes have a common abstract superclass *Block* (in Fig. 7a).

The information about the type of document is stored in the portion of the document object model depicted in Fig. 7b. Specifically, each class of documents in a digital library is defined as an instance of the object *DocClass*. In fact, defining each class of documents as an instance of a meta-level class – *DocClass* – allows us to achieve a greater flexibility when the digital library needs to be updated (by adding/deleting a new class of documents, through the methods *InsertClass*, *DeleteClass* of the class *DigitalLibrary*). Moreover, as previously stated, each class of documents is associated with a set of meaningful types of logical objects, called logical labels, thus adding a new class of documents requires the introduction of a set of new logical labels. This is performed by creating a new instance of the class Attribute for each logical label (Fig. 7b).

Above the protected resource there is the digital library software (see again Fig. 6). It consists of five layers. The lower layers are those more related to the machinery used to implement the digital library service and ignore the semantics of the repository, since they do not need to know the format of the data stored in it. Specifically, the Network Layer allows remote access to the library via Internet, while the Operating System Layer makes available all the functions of the operating system and ensures that the users who require an access to the repository have proper access rights. In detail, it maps names of the users that issue a request to the proper locations by means of the Name Server, and limits each user to what the administrator of the digital library (library's custodian) permits by means of the Authorization Server.

The layer of the Resource Managers mainly deals with the management of documents, viewed at different levels of abstraction. The lower box in this layer – ObjectStore DBMS – is a database management system. The upper box contains the Document Storage Subsystem, that is to

say, the document storage and access software. It is involved in both storing and retrieving items to and from the library collection, and updating and searching the library catalogs. It is worth mentioning that its scope is limited to aspects that are independent of the meaning and the internal representation of information items in the digital library. It is implemented as a client-server tool.

As we claimed in the introduction, within the aim of achieving the integration of heterogeneous DBs, the problem of overcoming the mismatches among the various DBMS technologies must be faced. Therefore, with the purpose of standardizing the system by freeing it from the specific query languages supplied by the current commercial products, instead of merely using the query language of ObjectStore, we developed a (meta-)query language based on first-order logic. It allows the user to formulate any query concerning the objects belonging to any collection in an instance of an object-oriented model as well as other kinds of DB models. The introduction of another level of standardization in the system, the query interpreter service, allows us to meet the aforesaid requirement with little effort. Indeed, the only additional requirement is a mechanism to convert the input queries, in the proposed language, into the native query language of the specific underlying DBMS.

The primitives of our query language are:

i) (<Object> IS IN <Class>)
ii) (THE <Attribute> OF <Object> IS <Value>)

where <Object>, <Class>, <Attribute>, and <Value> are metasymbols that can be properly replaced by variables or constants in each query. The former kind of primitives, called Class Query, allows us to answer membership class queries, such as "List all the documents in the class of scientific papers", or "Is the document #101 an instance of the class of business letters?", while the latter, called Attribute Query, is useful to find objects whose attributes meet specific conditions about their values. Examples of Attribute Queries are "List all the papers whose author is John Smith" or "Who are the authors of the paper #101?". A query is a proper combination of these two kinds of primitives through the logical operators AND, OR.

The productions, expressed in extended BNF, of the phrase structure grammar that generates the query language are as follows:

```
<Query> ::= <AndQuery> { + <AndQuery> }
<AndQuery> ::= <ClassQuery> { & <AndQuery> } |
               <AttributeQuery> { & <AndQuery> }
<ClassQuery> ::= (<Object> IS IN <Class>)
<AttributeQuery> ::= (THE <Attribute> OF
                        <Object> IS <Value>)
<Class> ::=  <ClassName> | <Variable> |
             <Pattern>
<Object> ::= <ObjectIdentifier> | <Variable> |
             <Pattern>
<Attribute>::= <AttributeName> | <Variable> |
               <Pattern>
<Value> ::= <Constant> | <Variable> |
            <Pattern> |  <Composite>
<Constant> ::= <Integer> | <Real> | <String> |
               <ObjectIdentifier>
<Variable> ::= @<String>
<String> ::= { <Printable> - [@, ", ?, *] }
<Pattern> ::= { <Printable> - [@, "] }
<Composite> ::= "{ <Pattern> | <String> }_0^N"
```

where:

<ClassName> denotes the name of a physical class in the database;
<AttributeName> stands for the name of a physical attribute of a database class;
<ObjectIdentifier> is the object identifier (OID) associated to an object;
<Printable> represents any printable character;
Two or more strings of the kind <Pattern> or <String>, present in <Composite>, have to be separated by blank spaces or tabulations.

The layer of the Application Enablers makes available several functionalities to the different users of the library, and hides operating system and machine differences.

The lower level of application enablers is the Learning Server, that provides a suite of learning systems that can be exploited concurrently by multiple clients in order to perform document classification and document understanding. As mentioned in Sect. 3, the application of these tools is twofold: both to recognize patterns and layout/logical structures of the documents in the library, with the aim to create search indexes automatically (Esposito et al. 1993), and to analyze end users' interactions in order to set the default query of the system according to their most frequent queries.

At the upper layer there are other tools that may be useful for the different library clients. The Mailer enabler implements a standard electronic mailing system. The Document Manager is in charge of helping end users with their special kinds of documents, mainly as regards their presentation and manipulation. The Query Interpreter is the inference engine that allows the user to formulate any query concerning the objects in the library by a first-order logic language.

The Browser enabler is a tool that allows the user to navigate into the digital library. It is intended to be exploited by people who do not know the organization of the library. It produces on-the-fly an HTML file corresponding to the document required by the user. The document bitmaps undergoing the preprocessing phase have huge dimensions, therefore they would not be effectively stored/managed/transferred in the IDL. Moreover, by using just bitmaps it would be impossible to "interact" with the text, in order, for instance, to perform a finer search in it as well as to create hyperlinks among parts of the same document or of different documents. This

calls for the construction of an HTML generator module, called *HGENE*, to automatically reproduce a stored document as an HTML file, needing much less storage space than the bitmap counterpart, and keeping the original nature of each part.

It is required that the generated HTML document be an accurate replica of the original bitmap document as to the layout structure. Thus, the document appearance should be rearranged to reflect the original one independently of factors such as the size of the window and the browser utilized. This consideration holds in case the user wants to print the document, too. Finally, the module must be able to cope with the potential change in class and/or attribute definitions by the librarian. The information for carrying out all these tasks is stored in the ".lay" file coming out from the document pre-processing phase with WISDOM. So, what HGENE really does is taking such files and extracting the data needed to reconstruct the document layout.

The Folder enabler is used to create new folders, add a document to a folder, and delete an existing folder. The Editor Enabler is activated when a user wants to change a document. This is possible on local copies of a document, unless the user is the library's custodian.

The Interface Layer implements the applications that actually interface the users of the library. Currently, a unique GUI based on any Web browser, allows the three categories of IDL users to:

- create/delete a digital library (IDL administrator);
- manage a specific digital library, provided that he/she possesses the proper access rights (library's custodian, or librarian);
- choose a specific digital library and query/browse it on the basis of the content of its documents (end user).

The GUI is designed around a state-transition model, with each state representing an HTML page. All the HTML pages are dynamically generated by Common Gateway Interface (CGI) scripts in C language in order to reflect the current content of the repository.

5 Management and usage

There are three different kinds of persons who can interact with IDL, in order to modify its content or just to query it (according to the role that they play in it and to the access rights owned): the Library Administrator, the Library Custodians (or Librarians), and the End Users.

At the top level of this hierarchy we find the Library Administrator, who is unique and whose fundamental task is that of supervising access to the various libraries involved in the system. A typical prerogative of the Library Administrator is the possibility to decide the inclusion of a new library in (or the elimination of an already existing one from) the system of federate libraries.

Each library involved in the system is managed by a Library Custodian, who previously received the proper password from the Library Administrator. Indeed, the access rights owned by a Library Custodian allow him/her to modify the content of the library he/she manages, by adding, deleting or updating the classes of the documents in the library and the related search indexes (attributes of the classes), as well as the documents themselves, which constitute the instances of the previously mentioned classes.

The End User is any person who has access to IDL via Internet in order to query it for the documents he/she needs and to see them in digital format, if it is the case. Each new user of a library (that is, each user who interacts with IDL for the first time) is asked to enter his/her own data, and then receives an identity code to be used in any new access in the future.

After having defined the role that each of the abovementioned figures plays in the context of the Intelligent Digital Library service, let us show a typical session with IDL by the Library Custodian and the End User. We omit showing a Library Administrator session, since his/her functions are mainly managerial ones.

5.1 The Library Custodian

Any time a new library is included in the IDL system, the Library Administrator assigns a personal password to its custodian. From then on, the librarian will have to enter it in a specific field of the homepage of the IDL (Fig. 8) in order to act on the content of that library. The system takes care of validating the password and, if the control succeeds, gives him/her the possibility of operating. Then, a menu is displayed, which includes all the possible kinds of actions he/she can perform.

If the insertion of a new class is chosen, the system displays the names of the classes already existing in the repository, and then asks for the name of the class to be added, the number of its attributes and the name of each attribute.

If the custodian wants to add a new attribute to an existing class, he/she must choose that class and then, after displaying the previously existing attributes related to that class, he/she can insert the name of the new one.

Another possible choice is that of inserting one (or more) new documents acquired from different kinds of sources: from a text file containing data about the layout structure of a document (".lay" file), in the local host or in a remote one, or from a list of layout files, which can be newly created or already existing.

Conversely, the deletion of an existing document is also possible, as well as of a class attribute after choosing the class it belongs to, or even the deletion of a class of documents.

A list of inspection options follows, whose aim is that of summarizing the repository's content. It is possible to see the list of all the classes of documents contained in the repository, with the related attributes. The custodian can

Fig. 8. IDL Homepage

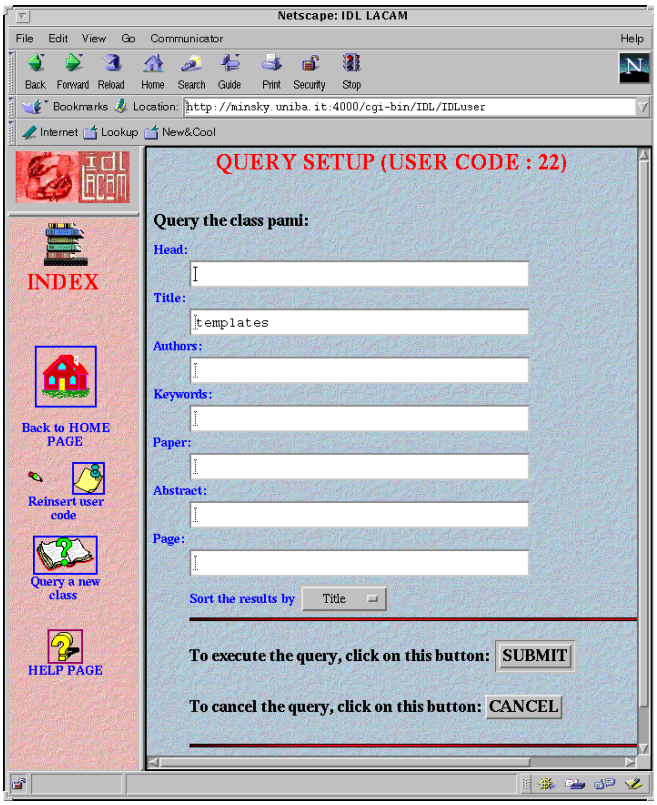

Fig. 9. Query submission

also have a description of the structure of each document in the repository.

The last option allows for a brief numerical description of the whole content of the repository, by means of a table.

5.2 The End User

The End User, too, can enter and query one of the system's libraries by just clicking on a button in the IDL's homepage (look back at Fig. 8). This causes a registration procedure to start, in which he/she can choose the specific library to be queried and enter the personal code identifying him/her in that library. In case he/she is a new user (i.e., he/she is consulting that library for the first time), a registration form is displayed, by which he/she can obtain the previously mentioned code after the insertion of his private data. Otherwise his/her personal data will be displayed. However, it is always possible to modify them.

When consulting a digital library, first of all, the user must choose the class of documents to be queried, among those included in the selected library. Then, he/she can submit the query (Fig. 9), which can involve any combination of attributes of the chosen class. For the sake of convenience, he/she can recall directly one of the last queries for each attribute, and decide the kind of ordering according to which the query results are to be displayed. They are presented in Fig. 10, and for each of them the user can also obtain the visualization of an HTML page (Fig. 11) representing the document itself. When the end-user is presented with the results of his/her query, he/she can find a *VIEW* button together with each retrieved document. This way he/she can get that document displayed on a window frame. One could easily notice that it is not a bitmap of the document, indeed it is an HTML page, generated on-the-fly by *HGENE*, the application enabler in charge of converting the internal format of the document into the HyperText Markup Language.

In the next section, it will be illustrated how the IDL logs the end users' sessions and in which way it exploits such a collected information in order to adapt the interface to the features that affect the interaction and to the needs of the end users.

6 Interface adaptivity

The aim of designing adaptive user interfaces is that of having a number of different solutions to match the variety and changeability of users, environments and tasks. In IDL a simple adaptive system has been implemented in order to produce a change in the output in response to a limited number of situations and user behavioral models.

A first level of adaptivity enables tailoring the interface, after a sufficient number of interactions, to the characteristics of the interaction of a specific user. A second level of adaptivity allows the system to autonomously

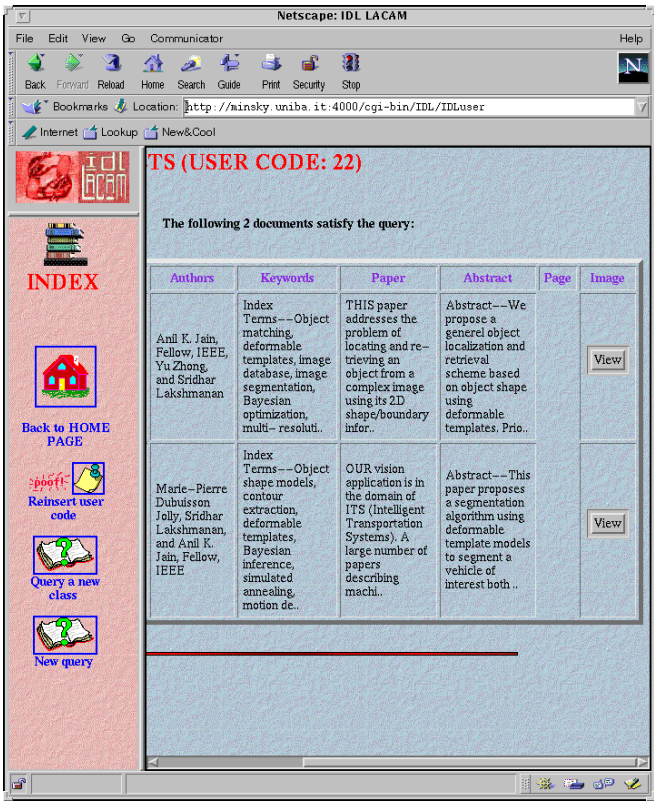

Fig. 10. Display of the query results

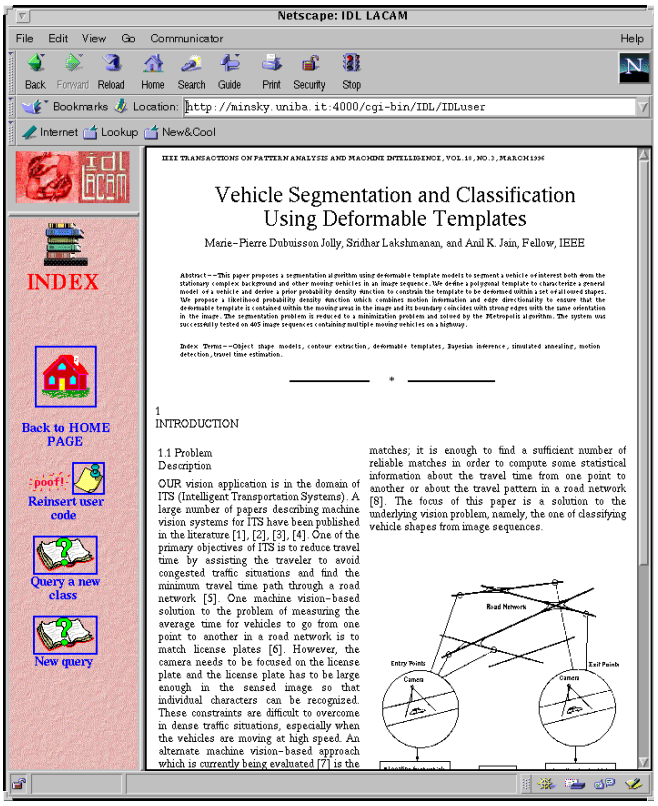

Fig. 11. Display of a retrieved document in HTML format

choose the kind of interface to present to the final user, once the class the user belongs to has been identified.

The rules which constitute the user classification patterns are inferred by the Learning Server from pre-classified examples of user interactions, as explained in Sect. 3.2.

As we pointed out above, each user of the IDL is provided with a personal identification code, represented by a number automatically generated sequentially by the system each time a new user connects to it. When a new user connects to the IDL, a registration form is presented in order to record his/her personal data. From then on, he/she will enter the system by just typing his/her own personal code.

Once the user has been identified (either as a new or as an already existing one), his/her behavior must be recorded in order to create/update the database about his/her queries, which will be used to set the default to the most frequently asked ones. It is important to note that such a database is influenced only by successful operations, since it would be nonsense to record operations which are impossible to carry out. Moreover, the information about a particular user is stored, but not used to modify the system default, until the user has made a certain number of interactions: only then will the data previously collected be processed. In the first interactions, on the contrary, the queried class and the ordering attribute will be set to the first available ones, and no value will be specified for the single attributes.

The information about each user is stored in a corresponding log file, that is updated after every successful query with the corresponding data. This operation is transparent to the user, which prevents him/her feeling "observed" by the system. For each performed query, this file reports the queried class of documents, the attribute chosen for ordering the query results, and a list of groups, each reporting the name of a document satisfying the query and the text of the query for each attribute, along with the information on the fact that it was validated (i.e., filled with a value by the user) or not (i.e., left empty by the user to indicate that its value was unimportant).

But just adding data to these files at every interaction would soon make them too large to be handled and analyzed. In order to limit the size of the stored data, without losing accuracy for the analysis, a file containing statistics about all the past interactions is used, whereas the log file is organized as a FIFO structure, remembering only the last queries at any moment. This is in agreement with the idea that the most recent queries better reflect the current needs of the user, while still taking into account the past ones by means of the previously computed statistics.

As soon as a user has reached the required number of successful interactions for the system adaptation to start, a new file is generated for him/her, containing some statistics. In particular, for each document class in the queried library, the related frequency in the last queries (updated at each new interaction) is reported, followed by the related attributes, each with the corresponding fre-

Fig. 12. Frequent query menus

quency as an ordering attribute and with the list of all the related query texts.

System adaptivity is obtained as follows: the default class to be queried is set to that with the higher frequency in the statistics file, and the ordering attribute is set to that with the higher frequency among those associated to that class. As to the query values for the various attributes, each field in the query form is followed by a list, initially empty (before starting the adaptation), containing all the past values queried for that attribute. The user can either specify a new value (which will have precedence over that chosen in the list), or choose one (or more) values in the list, so avoiding retyping them and allowing their composition with the AND/OR logic operators (Fig. 12).

Of course, none of the default choices made by the system forces the user to accept it, but he/she can change everything he/she wants.

In the IDL each new user is asked to fill in a form with personal data, and afterwards he/she receives an identity code - *User ID* - to be used whenever he/she enters the IDL again. Respectively, on the server side, the IDL Application Server associates each User ID with a *log file*, saving all the interactions of that user with the IDL.

Data concerning interactions are used to classify the user. User classification allows the system to autonomously choose the type of user interface that is presented to that kind of user. Currently, the adaptive environment of IDL implements three distinct user interfaces, namely a *form-based* interface, a *topic-map* based interface, and a *tree-based* interface (Costabile et al. 1998). The form-based interface is proposed to any member of the class *Novice*, while *Expert* users are assigned with the tree-based interface, and *Teacher* users have the form-based interface as their default. Of course, the user is allowed to switch to another interface at any time. When a user connects for the first time to IDL he/she is assigned to the default class, namely *Novice*.

After a user has been classified, the next problem to face is how to follow potential switches of the class the user belongs to, in case his/her behavior changes. It is plausible to foresee a *transitional* user, intended as a user becoming more and more skilled as he/she gets familiar with IDL. Moreover, this problem needs the system's ability to register and identify the user.

7 Conclusions, related work and future directions

Machine learning, together with intelligent object-centered techniques, can offer a valuable support when building intelligent digital libraries. Indeed, all the tasks related to information capture and semantic indexing can take advantage of the use of intelligent techniques and machine learning methods for layout analysis, document classification and understanding, while the integration of worldwide distributed digital libraries demands the definition of a standard query language for information retrieval. Moreover, machine learning techniques allow to infer user models from user interactions and this turns out to be useful to implement an adaptive interface. In the paper, we have presented a prototype of an intelligent digital library service proposing solutions to both the above issues.

It is hard to find work on adding machine learning techniques to digital library applications. Conversely, it is possible to find in the literature of information a number of methods and systems that exploit machine learning techniques to infer user profiles.

Specifically, systems like Syskill & Webert (Pazzani and Billsus 1997) learn and revise user profiles in order to determine which WWW sites would be interesting to a user. Such a task is driven by the specification of a topic and heavily relies on a Bayesian classifier. Letizia (Lieberman 1995) is a system that monitors users' behaviors when they are browsing the WWW, and tries to infer their interests. No explicit interaction with the user is required. WebWatcher (Armstrong et al. 1995) also monitors the users while browsing the WWW, so it can suggest which links to follow in order to reach a specific site from a starting one according to a specified goal. WebHound (Lashkari 1995) is an interactive system that requires each user to list the pages he/she is interested in, along with ratings for them. Then, the system autonomously detects other users that gave similar ratings and suggests

new pages to the user on the basis of the ratings by users with similar interests.

Future work will concern the extension of the digital library's tools and services in order to deal with different kinds of documents, such as topographic maps for applications like geographic information systems, and technical documents for applications that support project development.

References

1. Armstrong, R., Freitag, D., Joachims, T., Mitchell, T.: WebWatcher: A Learning Apprentice for the World Wide Web. Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments, Palo Alto, CA, 1995, pp. 6–12
2. Banyon, D., Murray, D.: Applying user modeling to human-computer interaction design. Artificial Intelligence Review. 7: 199–225, 1993
3. Ciardiello, G., Scafuro, G., Degrandi, M.T., Spada, M.R., Roccotelli, M.P.: An experimental system for office document handling and text recognition. Proc. 9th Int. Conf. on Pattern Recognition, Los Alamitos, CA: IEEE Computer Society, 1988, pp. 739–743
4. Costabile, M.F., Esposito, F., Semeraro, G., Fanizzi, N.: An adaptive visual environment for digital libraries. Submitted to Int. J. on Digital Libraries, Berlin, Heidelberg, New York: Springer-Verlag
5. Esposito, F., Malerba, D., Semeraro, G., Annese, E., Scafuro, G.: An experimental page layout recognition system for office document automatic classification: an integrated approach for inductive generalization. Proc. 10th Int. Conf. on Pattern Recognition, Los Alamitos, CA: IEEE Computer Society, 1990, pp. 557–562
6. Esposito, F., Malerba, D., Semeraro, G.: Automated acquisition of rules for document understanding. Proc. 2nd Int. Conf. on Document Analysis and Recognition, Los Alamitos, CA: IEEE Computer Society, 1993, pp. 650–654
7. Esposito, F., Malerba, D., Semeraro, G.: Multistrategy learning for document recognition. Applied Artificial Intelligence 8(1): pp. 33–84, 1994
8. Esposito, F., Malerba, D., Semeraro, G.: A Knowledge-based approach to the layout analysis. Proc. 3rd Int. Conf. on Document Analysis and Recognition, Los Alamitos, CA: IEEE Computer Society, 1995, pp. 466–471
9. Esposito, F., Malerba, D., Semeraro, G.: A comparative analysis of methods for pruning decision trees. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI 19(5): 476–491, 1997a
10. Esposito, E., Malerba, D., Semeraro, G., Ferilli, S.: Knowledge revision for document understanding. In: Ras, Z.W., Skowron, A. (eds.): Foundations of Intelligent Systems. 10th Int. Symposium, ISMIS'97, Charlotte, NC, October 1997. Lecture Notes in Artificial Intelligence, LNAI 1325. Berlin, Heidelberg, New York: Springer-Verlag, 1997b, pp. 619–628
11. Fisher, J.L., Hinds, S.C., D'Amato, D.P.: A rule-based system for document image segmentation. Proc. 10th Int. Conf. on Pattern Recognition, Los Alamitos, CA: IEEE Computer Society, 1990, pp. 567–572
12. Fox, E.A.: How to make intelligent digital libraries. In: Ras, Z.W., Zemankova, M. (eds.): Proc. 8th Int. Symposium on Methodologies for Intelligent Systems. Lecture Notes in Artificial Intelligence, LNAI 869. Berlin, Heidelberg, New York: Springer-Verlag, 1994, pp. 27–38
13. Goñi, A., Mena, E., Illarramendi, A.: Querying heterogeneous and distributed data repositories using ontologies. Proc. 7th European-Japanese Conference on Information Modelling and Knowledge Bases (IMKB'97), Toulouse, France, May 1997. IOS Press (Forthcoming)
14. Lashkari, Y.: The WebHound Personalized Document Filtering System. `http://rg.media.mit.edu/projects/webhound`, 1995
15. Lieberman, H.: Letizia: An agent that assists web browsing. Proc. Int. Joint Conference on Artificial Intelligence, Montreal, Canada, August 1995, pp. 924–929
16. Malerba, D., Semeraro, G., Bellisari, E.: LEX: A knowledge-based system for the layout analysis. Proc. 3rd Int. Conf. on the Practical Application of Prolog, 1995, pp. 429–443
17. Malerba, D., Esposito, F., Semeraro, G.: A Multistrategy Approach to Learning Multiple Dependent Concepts. In: Nakhaeizadeh, G., Taylor, C.C. (eds.): Machine Learning and Statistics – The Interface. New York: John Wiley & Sons, 1997a, pp. 87–106
18. Malerba, D., Esposito, F., Semeraro, G., De Filippis, L.: Processing paper documents in WISDOM. In: Lenzerini, M. (ed.): AI*IA 97: Advances in Artificial Intelligence, Lecture Notes in Artificial Intelligence, LNAI 1321, Berlin, Heidelberg, New York: Springer-Verlag, 1997b, pp. 439–442
19. Michalski, R.S.: Pattern recognition as a rule-guided inductive inference. IEEE Transactions on Pattern Analysis and Machine Intelligence 2(4): 349–361, 1980
20. Moustakis, V.S., Herrmann, J.: Where do machine learning and human-computer interaction meet? Applied Artificial Intelligence 11: 595–609, 1997
21. Pazzani, M., Billsus, D.: Learning and revising user profiles: the identification of interesting web sites. Machine Learning 27: 313–331, 1997
22. Quinlan, J.R.: C4.5: Programs for Machine Learning. San Mateo, CA: Morgan Kaufmann, 1993
23. Schlimmer, J.C., Fisher, D.: A case study of incremental concept induction. Proc.5th Nat. Conf. on Artificial Intelligence, Philadelphia, PA: Morgan Kaufmann, 1986, pp. 496–501
24. Semeraro, G., Esposito, F., Fanizzi, N., Malerba, D.: Revision of logical theories. In: Gori, M., Soda, G. (eds.): Topics in Artificial Intelligence. Lecture Notes in Artificial Intelligence, LNAI 992. Berlin, Heidelberg, New York: Springer-Verlag, 1995, pp. 27–38
25. Semeraro, G., Esposito, F., Malerba, D., Fanizzi, N., Ferilli, S.: Machine Learning + on-line libraries = IDL. In: Peters, C., Thanos, C. (eds.): Research and Advanced Technology for Digital Libraries, Lecture Notes in Computer Science, LNCS 1324. Berlin, Heidelberg, New York: Springer-Verlag, 1997a, pp. 195–214
26. Semeraro, G., Esposito, F., Malerba, D., Fanizzi, N., Ferilli, S., Lops, P.: IDL: a prototypical intelligent digital library service. In: Lenzerini, M. (ed.): AI*IA 97: Advances in Artificial Intelligence, Lecture Notes in Artificial Intelligence, LNAI 1321. Berlin, Heidelberg, New York: Springer-Verlag, 1997b, pp. 447–450
27. Shih, F.Y., Chen, S.-S.: Adaptive document block segmentation and classification. IEEE Transactions on Systems, Man, and Cybernetics - Part B 26(5): 797–802, 1996
28. Tang, Y.Y., Yan, C.D., Suen, C.Y.: Document processing for automatic knowledge acquisition. IEEE Transactions on Knowledge and Data Engineering 6(1): 3–21, 1994
29. Utgoff, P.E.: Incremental induction of decision trees. Machine Learning 4(2): 161–186, 1989
30. Utgoff, P.E.: An improved algorithm for incremental induction of decision trees. Proc. 11th Int. Conf. on Machine Learning, San Francisco, CA: Morgan Kaufmann, 1994
31. Wang, D., Srihari, R.N.: Classification of newspaper image blocks using texture analysis. Computer Vision, Graphics, and Image Processing 47: 327–352, 1989
32. Weinstein, P., Alloway, G.: Seed Ontologies: growing digital libraries as distributed, intelligent systems. Proc. 2nd ACM Digital Library Conference, Philadelphia, PA, July 1997 (1997)
33. Weinstein, P., Birmingham, W.P.: Service classification in a proto-organic society of agents. Proc. IJCAI-97 Workshop on Artificial Intelligence in Digital Libraries, Nagoya, Japan, August 1997
34. Wong, K.Y., Casey, R.G., Wahl, F.M.: Document analysis system. IBM J. Research Development 26(6): 647–656, 1982