Towards an Interactive Document Structure Recognition System

Frédéric Bapst
Rolf Brugger
Rolf Ingold

April 3, 1995

Abstract

This paper presents a new research theme at our institute in the field of document engineering; it describes a new general approach for document structure recognition and transformation. Several kinds of applications are presented, showing the great interest of such tools. The project that is described deals with three major research topics: co-operative recognition strategies in a multi-agents environment, automatic inference of a well-defined knowledge base, and high interactivity. The paper shows the problems detected so far and gives some main ideas currently discussed in our research group.

1 Introduction

The need of strongly structured documents has been recognized for a long time and their importance increases with the development of new software applications. Also document structure recognition has been investigated, in order to extend the OCR capabilities.

Automatic document analysis and recognition has been a major topic in our research group at the Institute of Informatics of the University of Fribourg (IIUF). Our experience has shown clear limitations of such approaches that can be characterized as follows:

- recognition results are subject to errors that have to be corrected manually;
- structures have to be specified for document classes accordingly with the application needs;
- structure models have often to be extended to take into account particularities of a specific document.

We believe that only interactive tools can overcome these lacks and increase the productivity of document structure recognition. Therefore our objective is to develop a new system architecture that achieves that goal.

Some people predict that the needs for document recognition will decrease in the future, since most documents will already exist in a machine readable form. This may be true in some sense, but we believe that the problem of structure recognition will remain for several reasons:

- 1. Although a document may exist in an electronic printable form like PostScript, the problem of converting it back to an editable form, for instance LaTeX, will still remain. In this case, we can think about that problem as the recognition of an uncorrupted image produced by a PostScript interpreter.
- 2. Structures can be considered in different ways; this means that a given document structure designed for one particular application may be irrelevant for another one. In that case, the problem is about automatic structure transformation. We believe once more that an interactive environment, working on a document image, may be the adequate tool to perform that work.

This paper presents a new project on Cooperative & Interactive Document Reverse Engineering (*CIDRE*). The main principles we want to develop in that context are:

- user interaction in order to specify, verify and correct document structures;
- inference of generic document structures and their use for automatic recognition;
- distributed multi-agents architecture.

Chapter 2 presents three possible applications that could be handled by a system as described above; the document models as well as a so thought realistic scenario for the user session will be discussed. Some current ideas about the knowledge base and the inference methods are described in chapter 3. Finally, chapter 4 sketches out some choices for the multi-agents approach.

2 Case Studies

The objectives presented so far may appear as too abstract. In order to facilitate further explanations, we want to introduce three different but very typical examples of possible applications for an interactive document structuring environment.

2.1 Construction of a Structured Bibliography

Let's consider a researcher who maintains a personal database of bibliographical references. After having read a book or an article he may wish to integrate some of their references into his own database. In order to have optimal access to the stored data it is necessary to transform them into a standardized format like BibTEX or SGML. Therefore it would be useful to have a system that is capable to recognize not only characters (the OCR task) but also the logical structure of a reference. Figure 1 shows some parts of a scanned bibliography.

Ackley, D. H. (1985). A connectionist algorithm for genetic search. *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, 121–135.

Antonisse, H. J., & Keller, K. S. (1986). Dynamic evaluation of imprecisely specified knowledge. *Proceedings of the Digital Avionics Systems Conference*. 596–600.

Antonisse, H. J., & Keller, K. S. (1987). Genetic operators for high-level knowledge representations. *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, 69–76.

Figure 1: Extract of a bibliography.

2.1.1 Document Description and Data Structures

One can distinguish two levels in the structure of a document — the microstructure and the macrostructure. The microstructure describes relations between low level entities like characters. In contrast to this the macrostructure can be viewed as the relationships and the global appearance (shape, size, typographical gray) of higher level entities like paragraphs, list items, lists or other textual blocks.

A bibliographical reference consists of several entities like *author*, *title*, *publisher* etc. As the important information of bibliographical references lies in the entities themselves and in the relationship between them, we should, as pointed out before, concentrate on the microstructure of the references. The internal structure of such entities (the content) as well as the relations between them (the context) can be described by a context-free grammar. Such a description of a class of objects, which is, in our case the class of references, is also called the *generic logical structure*. An extract of the grammar describing bibliographical entities from Figure 1 is shown in Figure 2.

The appropriate representation of the logical structure of one specific reference is a tree.

```
reference   ::= refhead '.' refbody
refhead     ::= authors date
authors     ::= author | authorlist
authorlist  ::= {author ','} author ',' '&' author
author      ::= aname ',' fname '.' {fname '.'}
date        ::= '(' year [',' month] ')'
refbody     ::= book | article | phdthesis | inproceedings | inbook
inproceedings ::= title booktitle ',' pages '.'
pages       ::= number '-' number
year        ::= number
number      ::= {digit}
fname       ::= 'A'|'B'| ... |'Z'
aname, month, title and booktitle ::= {character}
```

Figure 2: Extract of the generic logical structure of a bibliographical reference.

Figure 3 shows as example, how the first entry of Figure 1 can be represented according to the grammar of Figure 2.

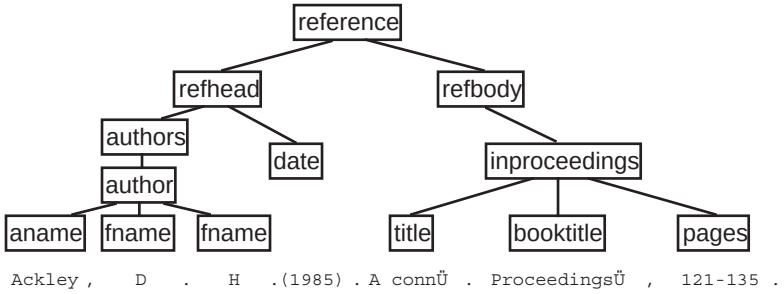

Figure 3: Extract of the specific logical structure of a bibliographical reference.

2.1.2 A Scenario of the Interaction with the System

The aim of our *CIDRE* project is to obtain a system that, on the one hand interactively tries to find a generic logical structure for a class of documents, and on the other hand interactively learns to match a specific entry to the generic logical structure it belongs to.

Let's consider how the interaction between user and system can be imagined:

As the system's knowledge base is empty, it cannot recognize or classify anything.

The user selects some parts of references and labels them (Fig. 4).

According to the labeled parts, the system tries to infer rules like:

- An author stands at the beginning of a reference.
- First names are capital letters followed by a period '.'.
- Booktitles are printed in italics.

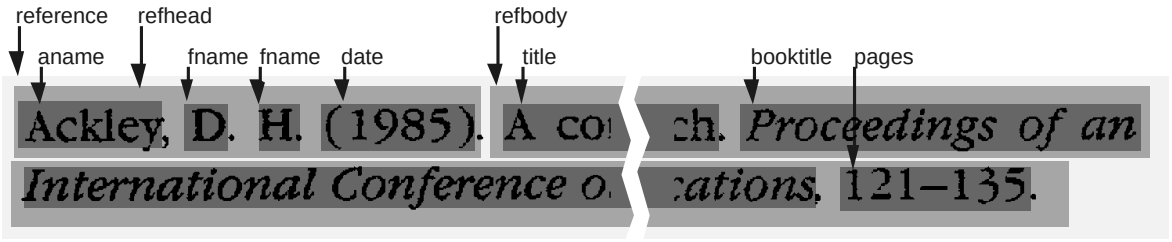

Figure 4: In a first teaching session the user has to label some logical entities.

- Page ranges consist of two numbers separated by one hyphen.

These rules are then applied to the next part of the scanned image. The system correctly labels parts like *date*, *pages* or *booktitle* but it can't classify the second author because it doesn't stand at the beginning of the reference (Fig. 5).

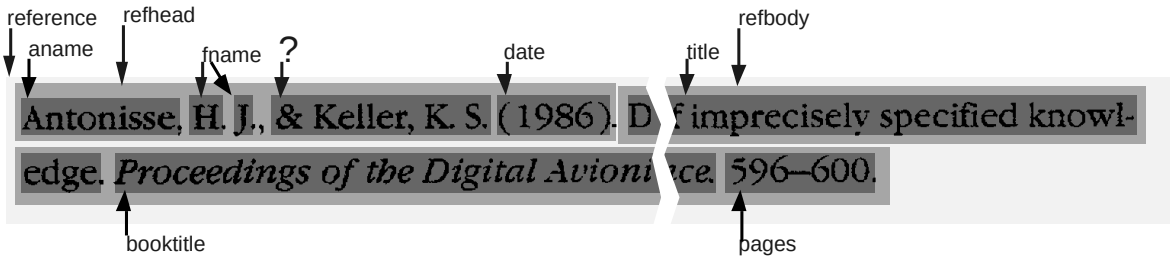

Figure 5: The system applies the previously learned rules to the next reference. Some parts of the reference could not have been classified correctly.

The user corrects the system's proposals by labeling the second author (Fig. 6).

The system now adapts the previously inferred rule for author classification:

- an author stands at the beginning of a reference or is prefixed by an ampersand '&'.

As we see in the example above, the user labels entities of the reference according to its specific logical structure. One labeled reference entity corresponds exactly to one node in the tree that represents the specific logical structure like in Figure 3. An important feature of our system resides in the fact, that the user is not forced to label entities top-down (from the root to the leaves) or bottom-up. He may begin to label entities that correspond to nodes in the middle of the tree. The system always tries to infer rules as good as possible.

Moreover he doesn't need to label entirely the learning set or, in other words, there is no need to use all possible labels at least once. He may for example systematically leave out all top level labels that would group labels at a middle level. In fact, there is no disadvantage to feed the system with incomplete learning data. For a user it may be interesting to get only

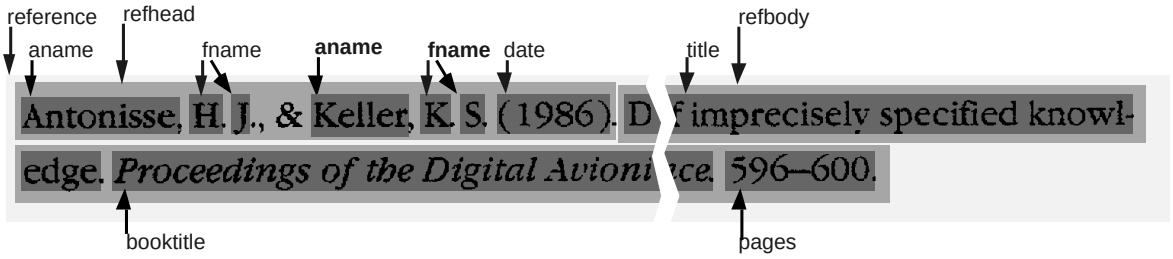

Figure 6: The user adds the correct labels to the reference which will serve the system to enhance the incomplete rules.

a part of the document structure. In the example of recognizing references we may think of a document in which the references themselves are arranged in a structured way. A user that only needs a linear list of references would like to ignore the structure of how references are organized. So he would only teach the system to recognize the internal structures of the references and not their external structure.

2.2 Building On-Line Help Facility

The second application example concerns the recognition of reference manual pages. We begin by explaining the reasons of this choice; then we describe precisely the structures involved in a concrete sample. Finally, we propose a typical session, as a mix of user intervention and system inference.

2.2.1 Motivations, Purposes

In this case study, the practical application centers on collecting information in order to build an on-line help facility about a specific domain. We suppose that the basic information is already available, but in a poorly structured form. The question is to restore it in an adequate way that will allow a convenient access. We can imagine a first situation where the data exist only in a paper form (user's guide, reference manual, etc.); this is the traditional field of structural document recognition via scanned images, and it includes the particular example discussed here.

One could remark that this kind of information is more and more provided on electronic support. But the need to transform it may still remain. For example, one of the most popular solutions to application documentation is the use of man-pages. These ones often have a rich and rather strict underlying structure, and are represented as formatted text. We believe that in such situations the use of an interactive environment working on images is the best solution to document re-structuring. In comparison with scanned images analysis, we can take advantage of the input richness (noise-free generated images, knowledge of the content in characters, indications about typographical attributes, etc.), but from the user's point of view, the task will be similar. The optimal use of interactivity is one of the *CIDRE* project challenges.

While the first case study main concern was micro-structure, this case study considers macro-structure aspects, i.e. the logical decomposition down to fragments (those entities which undergo the physical cut in lines). In general, of course, an interactive environment must transparently support both levels. Our goal is to show that the system should be able to learn a generic logical structure from scratch, i.e. without particular knowledge about the kind of document.

2.2.2 Sketch of a Model

We will now describe precisely our particular example, namely the Microsoft Excel Function Reference. It contains a list of Excel functions with indications about their effect, their syntax and other related information. Figure 7 shows a typical page taken from the original document and indications on its specific structure.

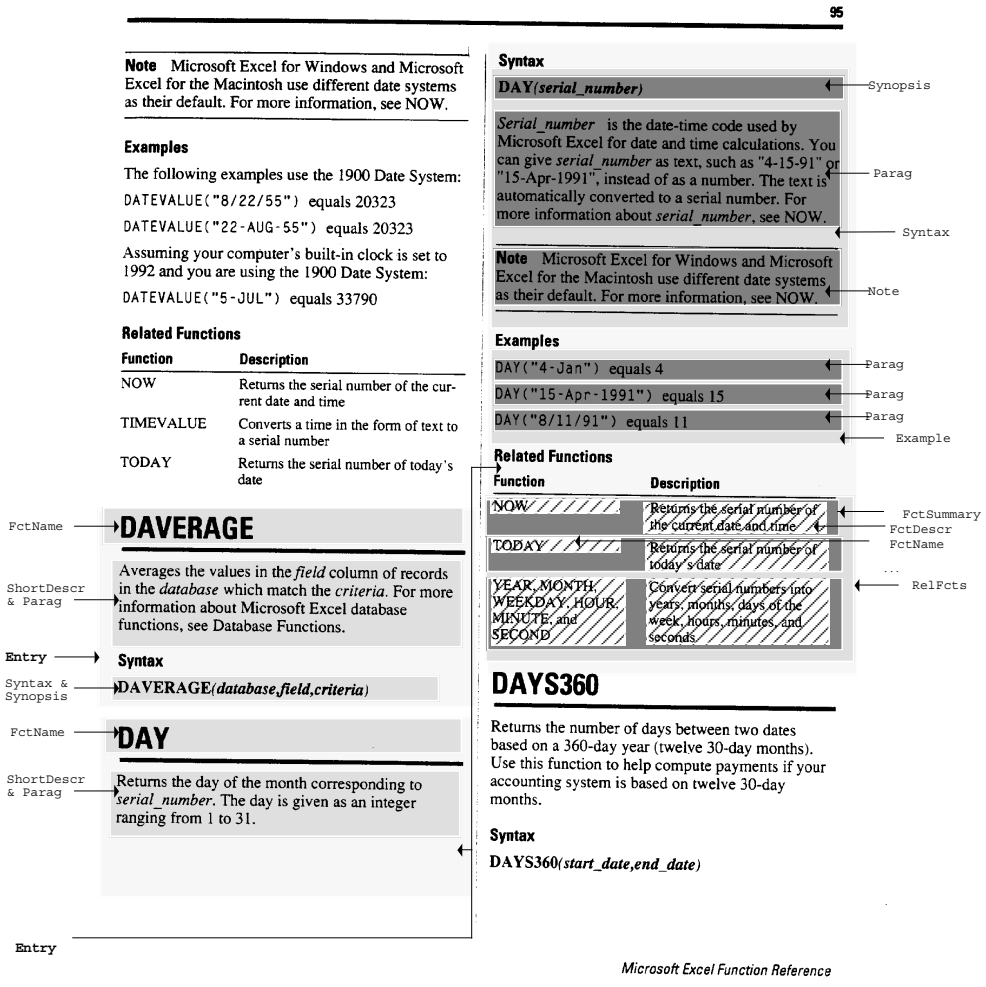

Figure 7: A sample of the Excel reference manual.

The system is expected to discover from scratch a model showing the hierarchical decomposition of each entry, which can be expressed by a formal grammar. In our example, Figure 8 represents the generic logic structure that has to be recognized. In general, it is the user responsibility to decide which entities are relevant according to the practical goal. In fact, Figure 8 gives only a rough approximation of the effective generic logical structure, but we assume that the user should not be forced to know exactly the complete grammar before starting a recognition session. In the *CIDRE* philosophy, the model is built incrementally through user interactions, so that the unexpected situations will be clarified when necessary. This schema is well suitable to handle structural exceptions that do occur in real documents.

In addition to the pure logical model, the system has to guess the presentation rules attached to each entity class. In Figure 9, we give an informal description of some important presentation rules for our example, as perceived by a human operator. Note that some physical entities (threads, dice) are pure presentation elements that do not match any logical entity, although they are important guides for recognition.

Both models (logical structure and presentation) will be used by the system to propose a matching between physical and logical entities.

```
Entry      ::= FctName ShortDescr Syntax [Remarks] [Example] [RelFcts]
ShortDescr ::= {Parag}
Syntax     ::= Synopsis {Parag | Enum} [Note]
Remarks    ::= {Parag | Enum} [Note]
Example    ::= {Parag} [Note]
RelFcts    ::= {FctSummary}
FctSummary ::= FctName FctDescr
```

Figure 8: A possible syntax for the logical structure of the reference manual.

```
FctName    :  bold font, size = 18 pts, before a fat thread
Syntax     :  after keyword "Syntax" (in bold font)
Remarks    :  after keyword "Remarks" (in bold font)
Note       :  between light threads, first word in bold font
Parag      :  interline = 4 pts
Enum       :  interline = 4 pts, indented, next to a dice
RelFcts    :  inter-column white separator
FctSummary :  interline = 4 pts
TheFct     :  first word in "all caps", in left column
TheSummary :  in right column
```

Figure 9: Some informal presentation rules for logical entities.

2.2.3 Progress of a Session

Let's try to imagine how to perform a complete recognition for this specific example, using the future *CIDRE* environment. We will describe a coherent scenario composed of user interactions and their expected effect on the internal behavior of the system. Thus both user interface and learning power are addressed.

Preparing the session The user starts a brand new session, indicates the page images and imports any useful model part of the knowledge repository (here nothing but universal notions of text, block, font, etc.).
The system could launch a feature extraction process.

Delimiting and labeling The user defines logical classes, by means of a few concrete examples. This is done by delimiting the corresponding image regions (e.g. a rectangle) and attaching them a logical label, as shown in Figure 10. Note that it is not a mandatory constraint to respect any ordering (e.g. top-down or bottom-up).
The system creates new logical instances and tries to build or update a model for their class. After the two first definitions, it would perhaps guess that the beginning of an `Entry` is always a word followed by a thread. When the `Note` is labeled, it modifies the model for `Entry` (to accept a `Note` as component) and updates the representation of the `Entry` instance containing this `Note` sample.

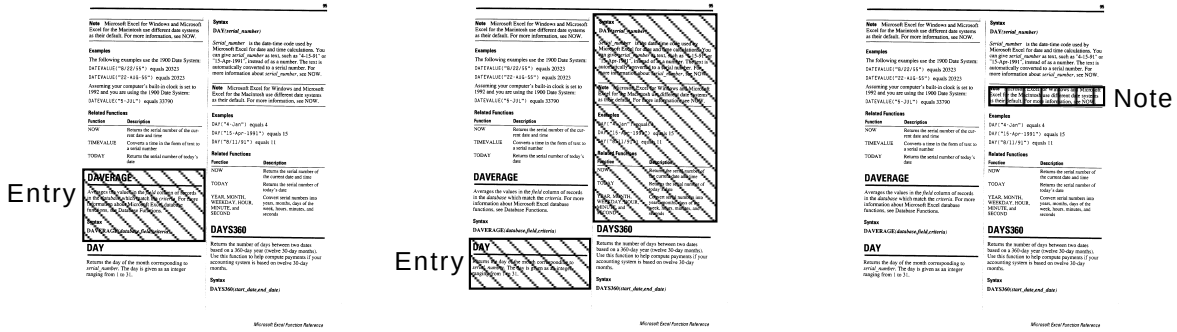

Figure 10: First steps in the session.

Asking for automatic entity search To verify whether the system has well modeled the `Note` class, the user delimits a region of the document (say the next page) and asks the system to discover a `Note` instance. The environment must provide means to visualize each system answer.
If the system fails to propose something, it may mean that it has inferred a bad model (too specialized features) and that additional examples are needed for generalization; otherwise, its answer will be either accepted or rejected interactively, giving rise to a reevaluation of the model.

Delimiting and labeling The user decides to label every entity in the current page. The operation of describing entity classes by naming a few samples can be iterated for every logical category. Note that this facility is not confined in a dedicated learning phase: the command is meaningful during the whole session.
After each labeling, the system updates all the models that are affected. Moreover, a track of every instance is kept, and the system guarantees that every instance still fits its class

model. It is hoped that the syntactic knowledge (context and decomposition of logical classes) converges towards the grammar of Figure 8 and that presentation knowledge can deduce recognition criteria looking like those in Figure 9.

Intensive recognizing The user wants now the system itself to drive structure recognition, choosing the validation and visualization policy. He can suspend this phase when it appears that some entity has been skipped and needs to be labeled manually.
The system passes through a certain region (by default the whole volume) and reports whenever it identifies any new entity, but without disturbing the user. During this phase, the models are mainly used to drive the recognition, namely the parsing and matching processes if we follow the classical approach [9]. But the knowledge may also be updated, especially at the time of invalidation. Thus, the inference engine is conceptually always in action, although we hope that the models will reach a stable state.

Models verification A first way to have an idea of the models quality consists in examining the results of the automatic recognition; the environment offers some facilities to visualize the recognized structure with several options (restriction to certain classes, differentiation between user-defined parts and system proposals, etc.), and a complete navigation command set.

Models edition As another means to supervise the models, an expert user can manipulate directly the knowledge base; a special command gives an overview of what has been learned. The expert user can thus verify and modify interactively the inferred syntax, presentation rules and discrimination criteria. Even though we do not exactly know for the moment how the knowledge base will be structured, we advocate this escape from the pure learning schema, both to evaluate the inference algorithms used and to save time when the user has in mind a precise description of a logical class. There is yet a lot of work to be done in order to define a clean protocol around the knowledge base management.

Results saving The user can save the session as a representation of the present state. If the recognition is over, he may save a formatted version of the document (e.g. in SGML). The models can be saved in the knowledge repository, or translated into a certain format (e.g. structure and presentation schemas for Grif [17]).

2.3 Mail Distribution in Office Automation

2.3.1 Motivation

Another illustration of a useful application resides in the field of office automation, where the postal mail is distributed through the network. In this context the incoming letter must be indexed according to the person it is addressed to.

Problems brought up by this recognition task concern the location of the address in the paper as well as the recognition of its internal structure, that is the name of the addressee and the department he or she belongs to.

2.3.2 Specific Problems

In opposition to the previous examples, we have to deal here with a great variability of layout structures (see Figure 11). First, there exist globally several different ways to organize the layout of a letter: the receiver's address may be located either on the right side or on the left side; it may appear below or above the date, and so on. Furthermore, even in the case of similar layouts, the location of addresses are not known with precision.

Second, the internal structure may differ in several ways: the addressee may be identified by his name or his function; or the letter may be addressed to a service or a team; sometimes the name is given on first line, sometimes it appears at the bottom line. The location, such as street or departments may be given with more or less details. Only the city name and the zip code can be considered as relatively standard.

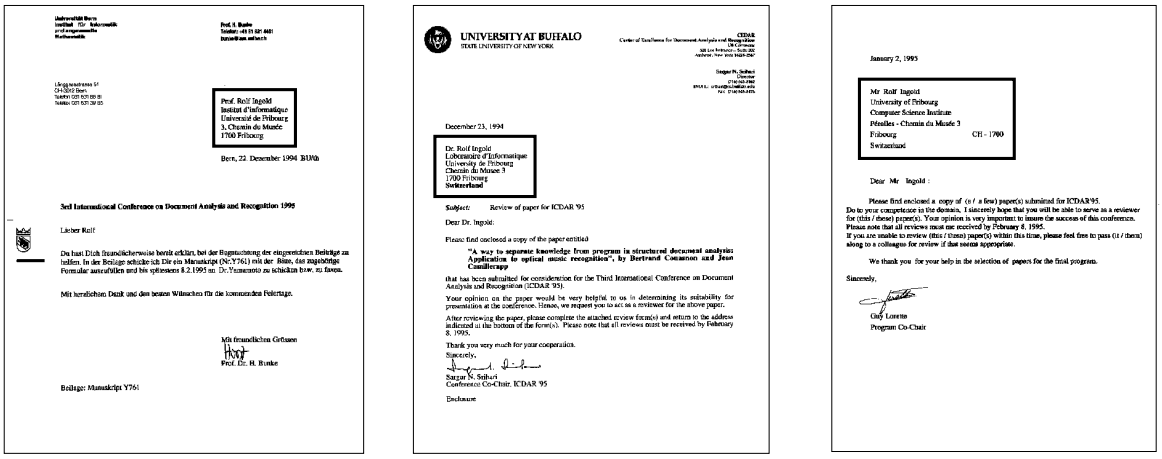

Figure 11: Layout variability for letters.

2.3.3 Knowledge Base Structure

We are now going to explore what kind of knowledge is required to achieve an intelligent mail addressees recognition system. In order to facilitate the discussion, we make several simplified assumptions, such as: only Swiss-style addresses are considered; the language is supposed to be French, and so on.

The fact that such strong limitations have been made illustrates the complexity of such a system and advocates once more for our interactive approach.

The necessary knowledge to recognize postal mail addresses includes the following information:

- a simplified and incomplete generic macro-structure for postal letters;

- a generic logical micro-structure of the address;
- several layout rules for address locations;
- several typographic presentation models for addresses.

The generic macro-structure could be represented as follows:

```
Letter   ::= ( Receiver [ Sender] [ Date ] [ Subject ] Content )
Receiver ::= AddressBlock
Sender   ::= AddressBlock
Date     ::= ...
Subject  ::= ShortText
Content  ::= LongText
```

This simplified model expresses that letters may contain two addresses, a date, a short text to describe the subject, and the content itself. Only the first and the last entity within this enumeration are supposed to be always present. It should be mentioned that the order in which these items appear has nothing to do with their location on the page; the date for instance can be located before or after the receiver's address, or even as a footer.

The generic micro-structure could look like that:

```
AddressBlock ::=( PersonName | Function ) Location City [ Country ]
City         ::= CityCode CityName
CityCode     ::= ['CH' ['-'] ] { DIGIT (=4)}
Person       ::= Title [Name]
Title        ::= ( 'Monsieur' | 'Madame' | Mademoiselle' | ...  |
                   'M' | 'Mme' | 'Melle' | 'Mr' | 'Dr' | 'Prof' | ...  )  ['.']
                   ['.']
PersonName   ::= [ Initial | FirstName ] LastName
Initial      ::= LETTER '.' [ ['-'] LETTER '.']
Country      ::= ('Switzerland' | ...)
Function     ::= SHORTTEXT
Location     ::= SHORTTEXT
FirstName    ::= SHORTTEXT
LastName     ::= SHORTTEXT
```

The most relevant entities that appear in this model are the `PersonName` and the `City` of the address. The former is supposed to have a strong structure in which first and/or last names can be extracted in order to compare it with an available personnel list. The description of the latter expresses the presence of a zip code, which represents one of the most relevant criteria for an address.

The layout rules are supposed to be verified by all known layout models. For instance, the width of the address block should not be larger than half of the page width; and it should contain between 3 and 7 lines.

The typographic presentation knowledge includes characteristics of address content, such as "the line containing the city can be bold or underlined".

3 Knowledge Representation and Acquisition

In this chapter we would like to discuss different aspects of the knowledge base we consider important for our project. In the first section we discuss the question what the knowledge base should be able to describe and give some ideas how these descriptions could be formalized. In the second section we raise the question about the knowledge base architecture and finally in the third section how the knowledge base could evolve.

3.1 Document Models

The importance of modeling document classes has been recognized in the early 1980's, and has then become a serious research subject. Complete document models should include logical as well as physical properties of documents.

The *logical structure* first defines the logical entities like paragraphs, lists, titles etc., that can appear in a document. Second, it describes the order in which these entities appear, for example titles are often followed by subtitles or paragraphs. And third, it describes how higher level logical entities can be decomposed in their lower level components like for example sections can be decomposed to subtitles and paragraphs. We illustrated these three aspects of logical structures for documents already in Figure 3 of Section 2.

The *physical structure* defines the entities that can appear in a printed or displayed document. Although physical entities are related to logical entities it is obvious that in general there is no clear one to one relationship between them. Some physical entities like headers or page numbers may exist without their logical counterparts or, logical entities like long paragraphs may be split off to several physical parts because they would not fit on one page. Nevertheless, there are cases, where one logical entity corresponds exactly to one physical entity such as titles or figures.

The structure extraction of one ore more documents becomes interesting when repetitive items like paragraphs, chapters or even entire documents are grouped together and reformulated. A more general structure can be extracted for such items by expressing the logical and physical aspects they have in common. This *generic* structure, also called the *document model*, is often subdivided into a logical and a physical part. Table 1 summarizes the notions introduced in this section.

	one specific document	**a class of documents (document model)**
composition of logical entities	specific logical structure	generic logical structure
presentational aspect	specific physical structure	generic physical structure

Table 1: Logical and physical structure of documents and document classes.

There exist two ISO standards for document models: SGML (Standardized Generalized Markup Language) and ODA (Open Document Architecture). Both standards have mainly been designed to simplify or even enable editing and interchanging of structured documents. In the SGML [21] approach, the actual text must be structured by adding markups, in order to delimit the logical entities of the text and thus to define its specific logical structure. These markups, where and how they can be applied to the text are defined in the document type definition DTD — the generic logical structure of the class. ODA [2] however, in addition to the generic logical structure also covers the generic physical structure of document classes. In contrast to SGML, ODA texts are stored and interchanged in the binary Open Document Interchange Format ODIF.

Some other document models, although not real standards, but parts of complete systems should also be mentioned here. LaTeX [12] and Grif [17] are both environments for the creation of structured documents. Especially Grif makes a clear distinction between the logical structure of a document and its layout.

Up to now, we discussed the role of document models in the context of document creation. However, document models would also be used for the inverse task of document recognition. Hu [9] for example implemented a system proposed by Ingold [10] that recognizes the text and structure of scanned documents. The document model used in their system — also called the document description — is able to describe the logical as well as the physical structure of documents. The generic logical structure is represented by production rules of a regular grammar. The generic physical structure is represented as attributes that are associated to the nonterminal symbols of the grammar. Table 2 illustrates the concept of attributed grammars applied on document structures.

	logical structure	**physical structure**
formalism	grammatical rules	attributes of logical entities
examples	`Chap` → `Tit` {`Parag`|`Fig`} `Parag` → {`Word`}	`Tit.Font = Roman` `Tit.Fontsize = 18pt`

Table 2: Document description with attributed grammars.

3.2 Knowledge Base Architecture

Since document structures become more and more complex, a considerable drawback of standards like SGML and ODA is their monolithical structure. As programming languages developed in the past 30 years from monolithic (FORTRAN, Cobol, PL/1) to modularized paradigms (Modula-2, Ada, C++), we postulate the same development for document descriptions. Especially the complexity of generally applicable document descriptions requires a well organized knowledge base.

Given for example a generic model for articles of one specific mathematical review, it would be necessary to copy the model entirely and modify some parts of it to adapt it to another mathematical review. This is the point where modularization of the models should be taken

into account. In order to avoid the rewriting of identical parts we can imagine to split up the description and organize it in a hierarchical way as it is shown in Figure 12.

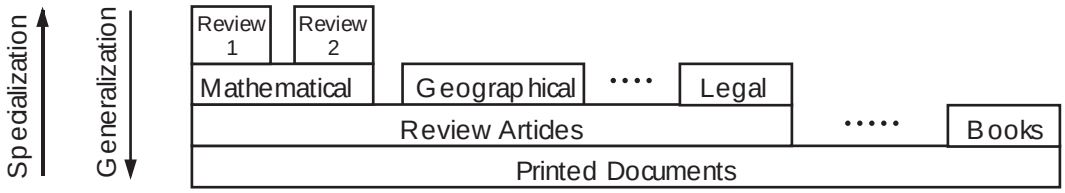

Figure 12: Organization of a hierarchical knowledge base.

Like in the object oriented approach, one module inherits all information from lower level modules and *reuses* their knowledge. The lowest base modules contain more general knowledge about documents whereas the higher ones contain specialized knowledge for specific domains.

There is another aspect concerning the knowledge base that should be kept in mind. As will be developed in the next chapter, we are planning to build the *CIDRE* system on a multi-agent architecture with cooperating and concurring agents or specialists. It is obvious that each agent may need its own part of the knowledge base that is adapted to its task. For example a font recognizer and a syntax checker would need specific knowledge. The consequence is that each agent or group of compatible agents may have its part of the database which, of course, is still modular and hierarchical as depicted in Figure 13.

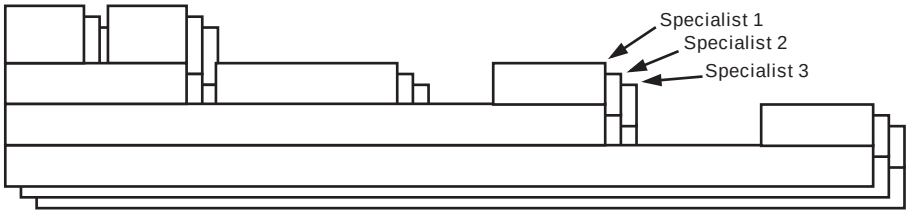

Figure 13: Different specialists may use specific parts of the knowledge base.

The problem of the knowledge base architecture is far from being solved and will be one of the important research subjects in the *CIDRE* project. One possible approach would be to extend an attributed grammar and to introduce an additional dimension for the different specialization or generalization levels. Figure 14 lists an example of such a grammar that describes the logical structure of mail addresses on letters. The description includes two abstraction levels: First, the 'Universal Document Structure' represents general knowledge that can be applied to almost all kinds of documents. It specifies for example the most general properties of text blocks, text lines or words. Second, the 'Mail Structure' represents specific knowledge about letters with sender and receiver addresses.

Considering for instance the universal knowledge that a `Page` can be decomposed into `Blocks` and `Lines` we see that this rule is not repeated in the corresponding item `HeadPage` that contains instances of `Address`, `Date` etc. The specific rules just indicate that a `HeadPage`

```
!  Universal document structures
Page       ::= { Block | Line }
Block      ::= { Line (>=2)}
Line       ::= { String.Word | String.Number | Char.Sign }
String     ::= { Char }
Date       ::= { Number.Day
               | ( @MonthName | Number.Month )
               | Number.Year
               }

!  Mail Structures
Letter     ::= Page.HeadPage
HeadPage   ::= ( Block.Address.Receiver
                 [ Block.Address.Sender ]
                 { Line.Others }
                 [ Line.Date ]
                 [ Block.Matter ]
                 Block.Content
               )
Address    ::= ( { Line }
                 Line.City
                 { Line }
               )
City       ::= ( String.Code Word )
```

Figure 14: Sketch of a formalism for modular grammars.

is a `Page` or that an `Address` is a `Block` and then they add information or they constrain existing rules. Consider for example the `HeadPage` rule: because we know that pages consist of blocks and lines, we know the same for `HeadPage`. This is the reason why only blocks and lines can appear in the `HeadPage` rule but the type and number of the blocks and lines has been specified more exactly. The first item in a `HeadPage` must be a `Block` which is further constrained to an `Address`.

We have seen in the example above, that there may exist ways to modularize grammatical rules. Of course, the formalism should be refined and completed and will be subject to further studies.

3.3 Knowledge Base Evolution

Until now we discussed what a knowledge base and especially a document model should be able to describe, what it contains and its possible architectures. A still open question is how such a knowledge base could be constructed. A crucial point is that the knowledge base cannot be built once off line and integrated to the system. As one of the *CIDRE* system objectives is a high interaction with the user, incremental learning techniques are needed. In this prospective article, we do not want to propose a solution to this non trivial problem, but we will discuss some related questions and try to extract the central difficulties.

The aim thus is to build step by step a modular knowledge base that eventually can be used by different agents or specialists. During a learning session, the system first gets some *samples* of a concept to be learned. Corresponding to section 2 these samples may be subtrees of the specific logical structure for the actual document. As sketched out in the scenario in an

idealized view, the system tries to understand the samples, applies them to yet unrecognized text and lets the user validate the proposals. By invalidating some proposals, the user passes counterexamples or *negative samples* to the system. Positive and negative samples are both valuable training information.

By analyzing the structure of the samples we can imagine that the system finds some rules that represent the samples in a more general way. This also means that the rules could describe a larger set of items than just the training samples. However, care should be taken that the rules do not integrate the negative samples. Good rules can describe many positive samples but are still precise enough to refuse the negative ones. Other rules that only describe a small subset should, if possible be modified or replaced by better rules. Sometimes however examples may appear that contradict well verified rules. In this case it would be better to leave the rule unchanged and to associate to it the contradictory sample as an *exception* to that rule. Exceptions are also positive samples, but they cannot be easily described by rules.

In fact, there exist two main operations on rules of a knowledge base to make it evolve: *generalization* and *specialization*. Generalization is the operation on the knowledge base that replaces a set of rules by one new rule, or that creates a rule being able to describe a set of positive samples. The new rule will then be able to describe at least the samples that were covered previously by the replaced rules, but normally the set of instances defined by the new rule increases. To illustrate this, think of the following scenario: A user labels four sections in an article that contain 2, 3, 5 and 9 paragraphs. The system may infer that all text blocks with exactly these numbers of paragraphs are article sections and all others not. In a next step, the system should replace the many rules by one that says: *a section contains n paragraphs,* $n \geq 1$.

The advantage of this generalization step is obvious. First , the new rule describes more adequately sections, because text blocks with 8 or more than 12 paragraphs would not have been recognized as sections before. Second, it is more economical to manage one rule instead of five. Less memory is occupied by the rule and the recognition of sections is more efficient.

The other operation on a knowledge base, the specialization works the other way round. It consists in adding rules or constraining existing ones to make a concept more precise. Specialization would be used in the case when two rules, that should represent different classes cannot be used to distinguish instances of them because of their inexactness. For example the rule *an item consisting of one text line printed completely in bold roman font* applies to titles and subtitles and can though not separate the two classes. Only when the sub rules ... *and font size 14 pt* or ... *and font size 20 pt* are added, the instances of the concepts title and subtitle can be recognized properly. In contrast to generalization the number of instances described by a new rule decreases on specialization and therefore there is no risk of including erroneously negative samples.

Let's quickly recall the four main dimensions of the knowledge base needed for the system: generic logical structure, generic physical structure, hierarchical modularization and modularization for different specialists. We will discuss now in short for each dimension, what learning techniques can be used for them.

Generic logical and physical structure: In the above example of rule generalization we showed the inference of the rule of a generic logical structure. The other example for

rule specialization concerned the generic physical structure. This illustrates that both generic aspects of document models could be generated by rule inference, provided that the selected formalisms are grammars.

Hierarchical architecture: Rule inference also adapts well to the hierarchical architecture of a knowledge base. We argued above that the constitution of the knowledge base can be viewed as the interplay of specialization and generalization operations on rules. Applied to hierarchical modules, this means that general modules would be more frequently modified by generalizations whereas specific domain modules often would be specialized. Nonetheless it will be difficult to find a general principle to decide in which knowledge module an actual rule adaption should be carried out.

Multi-agent architecture: Rule inference makes no sense, when the knowledge cannot be represented by a set of rules. So rule inference may not be the adequate learning technique for all specialists of a multi-agent system. A linguistic agent would perhaps prefer a dictionary for knowledge representation.

It is still an open question how rule inference can be accomplished in detail. One approach for grammar inference would be to annotate the given learning samples as production rules [18]. These primitive rules are an extremely specialized representation to exactly the training data. By recognizing repeated patterns with statistical methods, iterations or alternatives can be detected, which will serve to generalize the rules. The obtained rules can then be used to generate grammars.

Another method consists in integrating statistical information. In the example of generalizing the paragraph rules we in fact asked the questions:

1. Which features vary over the samples?
2. Which features are constant over the samples?

What varied in the sections was the number of paragraphs, but features like font or font size were constant. Features that vary on different instances of a class cannot be used to generalize rules and should therefore be removed from them. The inverse applies for specialization: those features that vary over instances of different classes should be used to distinguish them and are thus good candidates to be integrated into rules.

4 Multi-Agents Architecture

This chapter is devoted to the software architecture of the desired system. Actually, one of the main axis of the *CIDRE* project, along with the inference power and the strong interactivity, is the need of an implementation that reflects the cooperative aspects of the components. The field of multi-agents system has been explored for several years and is still intensively studied. It consists roughly in the conception of an application in terms of (more or less) autonomous working entities dived in a shared environment.

4.1 General Motivations

When an application is too complex to be implemented as a monolithic piece of software, it is often preferable to decompose it into some agents set.

One of the main advantages of the multi-agents approach is its ability to produce more maintainable coding. Once a convenient decomposition has been defined (and this is certainly not an easy step!), each agent is encapsulated in small independent packages that facilitate building, managing or testing, and the system is likely to be extensible without great efforts. This kind of modularity is particularly beneficent when different sources of knowledge might participate in the construction of a solution. Moreover, explicit separation between the agents' tasks and their interactions forces a clear specification of the agents competence, and promotes the testing of different collaboration strategies. Sometimes the skeleton of an agent can be defined as a reusable piece of code. Globally, the system is more flexible because it can be tuned to use various methods, either in the agents internal algorithms or in the coordination of their work.

Besides, the multi-agents programming can be seen as a first step towards an effective parallelization of the computing tasks. It is obvious that once an application has been designed as a set of autonomous living entities, it can be implemented by separate processes, which may then execute on different machines. This is not negligible when dealing with complex applications that request large computation power. Anyway, the evolution of the hardware clearly demonstrates that parallel machines and environment are tending to become unavoidable in computer science.

As well as the potential increase of performance, the decomposition into agents can be an insight to improve the system robustness. It is possible that some parts produce results that are useful but not essential for the whole application; or we can imagine that some components are deliberately duplicated and sent to different machines; in both cases the system will be more prepared to cope with a failure in the execution of the extra process.

Finally, it is very desirable to rely on multi-programming techniques when dealing with highly interactive applications. Indeed, the pure event-driven mechanism has some drawbacks if some actions last a long time. It is not acceptable for the user to be blocked on the accomplishment of some tasks; for an environment to be really user-friendly, the human operator must be able to control entirely the progress of a session. It can be interesting to bind the user behavior with a separate process that can influence the execution schema of other concurrent processes. We would like to evaluate this principle in the *CIDRE* project. The next section shows other arguments in favor of a multi-agents implementation for the specific problematic of document structure recognition.

4.2 Relevance of the Approach in Document Recognition

4.2.1 Competition between Algorithms

At several stages of the document recognition process, it is often the case that many approaches can be used to achieve a given task. In our institute for example, we have implemented many different algorithms to perform segmentation by ascending or descending methods [3], and we own a collection of OCR tools, either commercial packages, or self-implemented solutions (based on ternary mask matching, HMMs, neural networks, etc.) [5, 20].

Sometimes it would be interesting to apply various methods and compare their results in order to enhance the solution quality. We can imagine a kind of confrontation mechanism that eliminates unlikely proposals, either by simple election or with the help of complex heuristics. Such a design can easily be translated into a set of competing agents.

In general, different algorithms may have complementary properties in terms of efficiency, confidence rates or sensitivity to special conditions. A challenging issue for the system would be to choose the appropriate method according to the particular input being processed. In the OCR task for example, we could first use a fast algorithm on the whole page, and later apply locally an extended analysis on unrecognized parts.

With the multi-agents design, various strategies of competition between algorithms can be tested and the system could even be tuned dynamically.

4.2.2 Collaboration between Analysis Levels

The different tasks involved in document recognition are generally highly interdependent. Most often, several components are likely to participate in the resolution of the same problem. At a certain analysis level, it might be detected that the intermediate results used as input suffer from inconsistency. The relations between character segmentation and OCR constitute a typical example. This feedback phenomenon can be generalized by allowing every computed proposal to be criticized by any agent, and by organizing a conflict resolution strategy.

There are situations where the collaboration is an explicit key concept of the method. This is the case when combining an a priori font recognition with a mono-font character recognition. Suppose that the Optical Font Recognition (OFR) can attach distribution probabilities among an input set of fonts [22]. Suppose also that the OCR gives good results when the font is correct and leads to a high unrecognized rate otherwise. Then a line containing strings of different fonts can be recognized after a coherent sequence of a few iterations of both components, as illustrated in Figure 15.

By the way, note that the agents must offer a rich interface in order to be collaborative. In the traditional sequential schema, almost all individual tasks represent poor primitives without reasoning power; their specification must be extended to become agent-oriented. For example, an OCR agent is not only expected to return a stream of positioned characters from an image block, but could also be asked to propose several alternatives with confidence rates, or to evaluate and criticize an existing solution. A lot of work remains to be done in the design of agent interfaces and development of new algorithms.

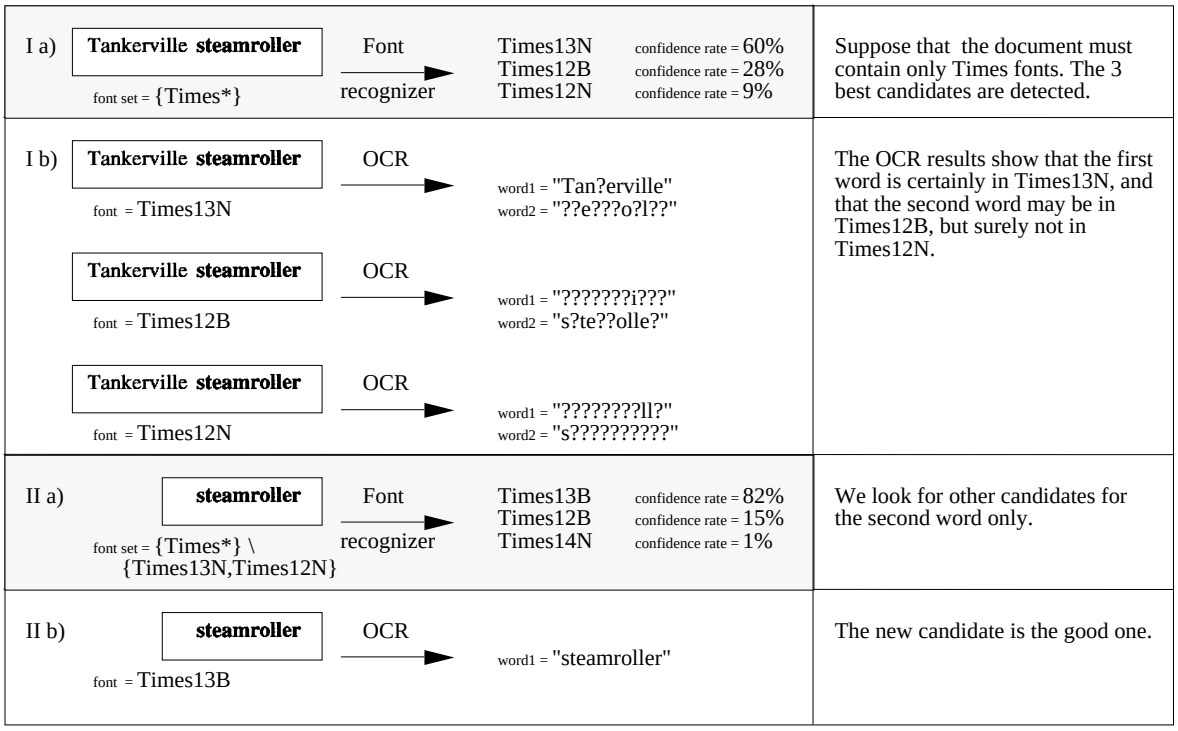

Figure 15: Intelligent collaboration between a font recognizer and a mono-font OCR.

Once the recognition task has been decomposed in a collection of small specific experts, the end-user can be considered as an exceptional expert of high confidence. Actually, nothing prevents from deeply integrating thus the user in the system design. This should have a great impact on the quality of human-computer interface.

Exploiting the collaboration is a central issue for the *CIDRE* project. We believe that document recognition is a strongly cooperative task distributed among many sources of knowledge, and that the multi-agents paradigm can especially help us for the design.

4.2.3 Parallel Nature of the Data

The recognition of a whole document involves a lot of operations concerning only a subset of the entire data collection. Although we need a global integration to produce a coherent final solution, the repetition of a same task on independent input samples can be observed at many steps:

- image preprocessing is often applied on single pages or blocks;
- feature extraction is also connected to bounded regions;
- the content in characters can be evaluated by applying OCR/OFR on partial text blocks;

- even structural analysis is likely to produce independent matching tasks, although it has to be controlled by a global parser.

This characteristic encourages a concurrent approach in the implementation. For example, we may run several specialists of the same nature (e.g. OCR) with different inputs. If the tasks are themselves independent (e.g. deskewing before noise reduction), it may be advantageous to design a pipelining protocol. Thus with multi-agents programming, a high level of background processing can be achieved under the control of user interactions.

4.3 Possible Architectures

The real difficulty in the application of the multi-agents style resides naturally in the modeling of the whole system in terms of autonomous agents and coordination schemes.

While the classical structured programming has given rise to important design skills, the advanced paradigms in computer science (parallelism, multi-agents, etc.) often suffer from a terrible lack of methodology. The programmer who wants to apply these new models for his specific problem has to study literature examples, and these are often either toy illustrations, or perfect final solutions, showing nothing of the design process. In fact, it is quite easy to see the defaults of a multi-agents decomposition, but the design of a good solution is generally a troublesome step (because badly mastered).

We will now sketch a few starting points to overcome the difficulties of designing a relevant architecture for the *CIDRE* project.

4.3.1 Emphasis on the Processing

A natural idea to choose an agent set is to take a closer look at the different tasks participating in the problem resolution. An exhaustive list of processing pieces involved is made up, and the designer analyzes it thoroughly in order to coherently group related elements. This approach is analogous to what is sometimes called the specialists-oriented parallelism [6].

Figure 16 presents an intuitive collection of interdependent tasks composing the *CIDRE* engine. We have placed them according to three main activities, namely the physical and logical recognition, and the models construction. All the management of the recognition session itself (GUI, file I/O, etc.) has here been summarized in a single process somewhat apart from the others.

This representation is clearly too poor because it does not exhibit the interconnections, and most important it does not address the control mechanisms to get the whole thing run smoothly and consistently. It is perhaps reasonable to define an agent for each of these tasks, but there will be certainly other agents in charge of driving them and coordinating their actions. We will focus on this part of the system in our further investigations.

4.3.2 Emphasis on the Data

As an alternative starting point, we can consider the problem from the point of view of the data being processed by the system. An analysis of the input information, the intermediate

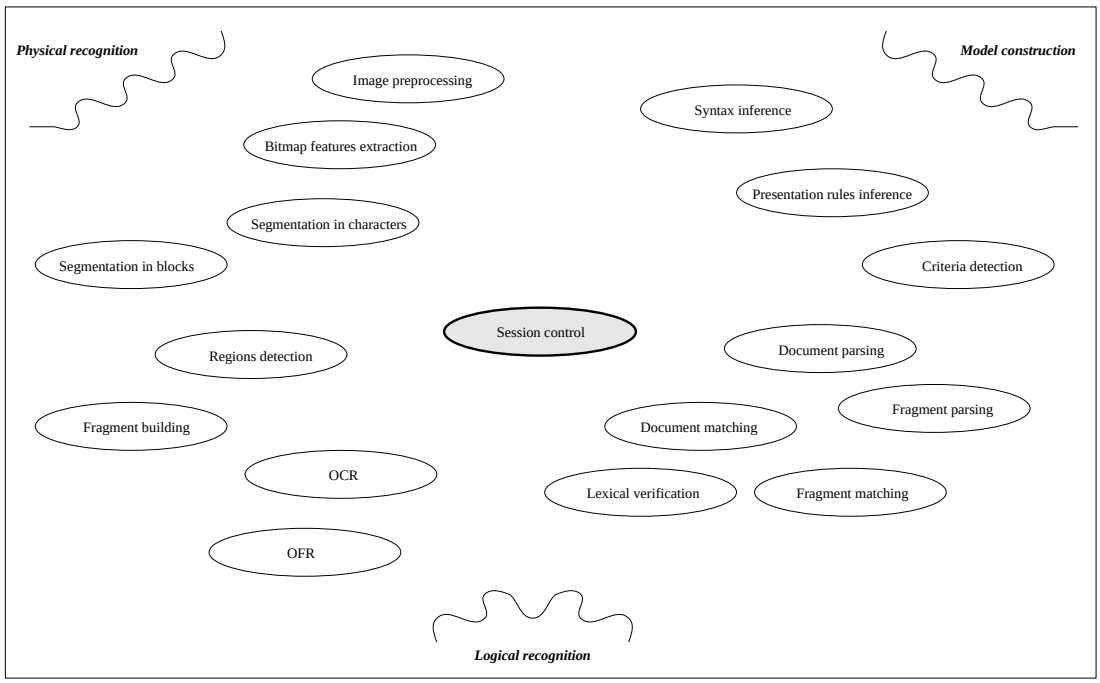

Figure 16: A representation of the tasks in the system engine.

results and the desired output serves as the basis of a model where the agents essentially match the data managed by the application. This technique is close to the paradigms of results-oriented parallelism and living data structures [6].

The *CIDRE* environment aims at both putting specific documents in a well structured form adapted to the user's requirements, and generating reusable models for documents sharing similar pieces of structure. So there are reasonable arguments to partition the information around two basic object categories: specific structure components and the generic entity classes.

Some important information manipulated by *CIDRE* is presented in Figure 17. Even though the whole database has not been clearly defined yet, this illustration gives an approximated overview of the system from the data perspective. It makes a first separation between class descriptions composing the document models, and the instances forming the specific structures. Each of these objects is characterized by its set of parameters. The specific structure constituents are specialized in physical and logical levels, with different respective attributes.

Once again this representation hides too many elements to find a direct architecture transcription. Among others, the relations between all these objects are not specified but only suggested. Furthermore, the evolution mechanisms are totally neglected. Despite of these present lacks, this point of view could lead to a coherent multi-agents decomposition.

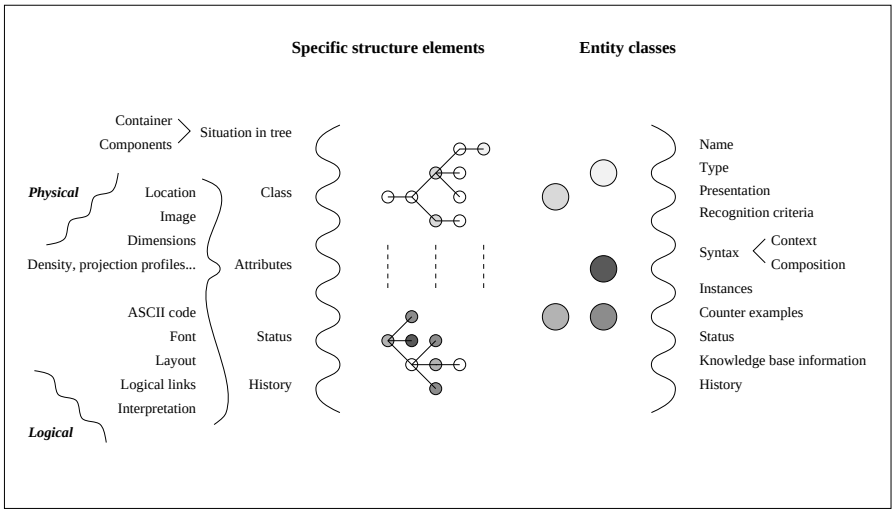

Figure 17: A representation of the information managed by the system.

4.3.3 Emphasis on the Relations between Knowledge Sources

In the two orientations proposed before, we concentrated on tasks vs data, and left the interconnections between system components for future investigation. But in a really cooperative architecture, these relations are of first importance, because they explain the coordinated evolution of a solution. So it is not foolish to begin the design by an extensive analysis of the various links tied through the whole stuff. This will hopefully help us to detect the crucial points where an agent definition can simplify the model, or make it more elegant. In our document structuring problem, interdependencies occur at different stages, and it is not trivial to classify them.

First, any piece of result acquires references to other system components, at the moment of its creation (e.g. input data used) and during its evolution; depending on the result kind, there are moreover logical links (e.g. class descriptor for a logical entity, neighbors for a physical block, etc.).

Second, a specialized task has a certain competence that positions it in the architecture. In fact its specification anticipates the potential uses in different circumstances, and list the affected kinds of data. Specialists may keep a history, and links to other related specialists. As an execution mechanism, we can define a set of operators that will activate an evolution of the solution space, such as adding a new result or criticizing an existing one.

Finally, the session management handles activities that suppose an access to any part of the system, e.g. global solution management, conflict detection among partial results or GUI management.

Such a point of view shows promising similarities between *CIDRE* and a project of the IDIAP research institute at Martigny [7], which aims at developing a general environment for cooperative treatment, and applying it to the problem of continuous speech recognition. Roughly, they propose a mixed architecture composed of micro-agents managing elementary results

that have to integrate with one another, and macro-agents implementing specific areas of expertise. All agents are reactive and the evolution mechanism is based on a negotiation protocol. We will certainly take advantage of this current research project, and are trying to find an adaptation of the model for our own application.

4.4 Implementation Platforms

The multi-agents paradigm seems to fit quite well the *CIDRE* philosophy. We will now focus on the programming tools that may be useful for a concrete implementation. All the platforms we mention below have still to be seriously evaluated.

The classical way to deal with concurrency consists in designing a set of sequential processes using message passing as communication protocol. These specialists are written in an imperative language like C or Ada, and may use a distributed platform like PVM [1]. This schema is getting more or less stable and supports a long tradition of structured programming. In our institute, we could moreover take advantage of an intensive study around new coordination concepts and related programming methodology [15]. On the other hand, there is an important programming overhead if the programmer wants to simulate distributed AI techniques because of the rather low level programming environment.

Concurrent logic programming stands for an appealing alternative: it offers high level mechanisms and a rich execution schema. The flat concurrent language Strand [8] has been intensively used in our institute. It provides very elegant solutions to process coordination, and brings the expressiveness of logic rules. But the code gets hard to maintain when the application to implement grows. In fact it lacks some syntactic facilities to enhance source readability. We are now looking more and more at Oz [19]; this language offers together many different paradigms, namely:

- strong mathematical underlying model,
- object-oriented facilities,
- constraint-based control,
- logic rules with guards,
- concurrent execution,
- open language,
- embedded graphical user interface tools.

All these features make Oz a serious candidate for writing (parts of) a first prototype.

As a completely different alternative to traditional platforms, and depending on the agent set granularity, it may be advantageous to think of the connexionnist approach, which is wider than pure neural networks. Michel Ludwig has defined a multi-agents reactive model where all interactions are described with charges and forces [14]. He found original solutions to a few classical problems (e.g. graph coloration or obstacles avoidance), but there is a lack of

hints on how to apply the model in general, as the programmer has to translate high level coordination into low level interactions. The structure recognition tasks are perhaps too complex to be modeled in this way, but some specific sub-problems like segmentation can be good candidates.

The distributed AI comprises also the study of multi-experts systems, and some environment have already been developed. Many of them rely on the blackboard notion. In this field there are several available platforms, such as the coordination language Linda [16], the blackboard manager GBB [4], or the multi-experts system Atome [11]. Lander describes a complete framework for multi-agents design based on GBB and called TEAM [13]. It addresses many fundamental topics such as collaboration, search strategies, negotiation, search operators (proposal, critic, extension, etc.), global solution space or solution quality. Even if the prototype is not really stable, this excellent discussion about clearly defined concepts will certainly guide us in designing the *CIDRE* architecture.

Finally we should have a closer look at the field of document composition, which is the reverse problem. We are particularly interested in the Grif environment [17] for several reasons:

- the clear and complete underlying model makes it one of the best structured document editors;
- it is provided with a full API that can be used for any application that deals with structured documents;
- conversions between standard formats (SGML, LaTeX) are supported;
- it offers an easy-to-use GUI library.

In fact, Grif could play the role of an integration platform for the *CIDRE* implementation.

5 Conclusion

In this paper, which is purely speculative, we have tried to exhibit a new research theme that seems to be innovative in the sense that it links two main research fields: document analysis and recognition, including OCR, and electronic publishing.

The objective of our project is to build an interactive environment that allows assisted document structuring. The tools we plan to develop should be useful either in controlling and correcting automatic recognition from scanned document images or in interactively transforming computerized document structures.

The research is still at an early stage, but our experience in recent work (document layout analysis, OCR and optical font recognition, document structure recognition, etc.) should be very useful. The main subjects on which we want to focus our activities concern three topics:

- knowledge base structure for document modeling;
- architecture of an adequate distributed multi-task environment;
- user-friendly interface interacting with automatic processing.

References

[1] A.Geist. PVM-3 user's guide and reference manual. Technical Report 12187, ORNL/TM, 1994.

[2] W. Appelt. *Document architecture in open systems: the ODA standard.* Springer Verlag, 1991.

[3] Antoine Azokly, Abdelwahab Zramdini, and Rolf Ingold. Reconnaissance de structures physiques de documents composites. In Abdel Belaid, editor, *CNED'92, Traitement de l'écriture et des documents*, pages 30–39. BIGRE, Juillet 1992.

[4] BBtech. GBB and Net-GBB. Technical report, Blackboard Technology Group, Inc., 401 Main Street, Amherst, Massachusetts 01002, 1994.

[5] Boris Bellorini. Extension d'un module de reconnaissance de caractères sensible à la fonte. Second cycle student project, IIUF-Université de Fribourg, December 1994.

[6] Nicholas Carreiro and David Gelernter. *How to write parallel programms: a first course.* MIT Press, 1990.

[7] Jean-Luc Cochard and Philippe Froidevaux. Environnement multi-agents de reconnaissance automatique de la parole en continu. In *3èmes journées francophones sur l'intelligence artificielle distribuée et les systèmes multi-agents*, Chambéry - St Badolph, France, March 1995.

[8] I. Foster and S. Taylor. *Strand - New concepts in parallel programming.* Englewood Cliffs, 1990.

[9] Tao Hu. *New methods for robust and efficient recognition of the logical structures in documents.* PhD thesis, IIUF-Université de Fribourg, 1994. Thesis no 1076.

[10] R. Ingold. *Une nouvelle approche de la lecture optique intégrant la reconnaissance des structures de documents.* 777, EPFL, Lausanne, 1988.

[11] H. Laasri and B. Maitre. Flexibility and efficiency in blackboard systems: Studies and achievements in ATOME. In Academic Press, editor, *Blackboards and Applications.* 1989.

[12] L. Lamport. *Latex, a document preparation system.* Addison-Wesley, 1994.

[13] Susan Lander. *Distributed search and conflict management among resusable heterogeneous agents.* PhD thesis, University of Massachusetts Amherst, May 1994.

[14] Michel Ludwig. *Conception et réalisation d'un langage de description d'interactions pour des systèmes d'agents autonomes.* PhD thesis, IIUF-Université de Fribourg, 1995.

[15] Oliver Krone Marc Aguilar and Beat Hirsbrunner. The COLA approach - a coordination language for massively parallel systems. Internal working paper 94-12, IIUF-Université de Fribourg, 1994.

[16] D. Gelernter N. Carriero. Linda in context. *Communications of ACM*, 32(4), April 1989.

[17] Vincent Quint. *Grif – Manuel utilisateur.* Imag - INRIA, December 1994.

[18] R. Schalkoff. *Pattern recognition, statistical, structural and neural approaches.* Wiley, 1992.

[19] G. Smolka. An Oz primer. Technical report, German Research Center for Artificial Intelligence (DFKI), January 1995. Draft version.

[20] Lukas Theiler. OCR with hidden Markov models and neural networks. Second cycle student project, IIUF-Université de Fribourg, July 1994.

[21] E. Van Herwijnen. *Practical SGML.* Kluwer Academic Publishers, 1990.

[22] Abdelwahab Zramdini and Rolf Ingold. A priori font recognition using a bayesian classifier. Internal working paper 94-04, IIUF-Université de Fribourg, February 1994.

Contents