INTEGRATED MULTI-AGENT ARCHITECTURE FOR ASSISTED DOCUMENT RECOGNITION

FRÉDÉRIC BAPST, ROLF BRUGGER, ABDELWAHAB ZRAMDINI, ROLF INGOLD
IIUF-Informatics Institute of the
University of Fribourg
Chemin du Musée 3
CH-1700 Fribourg (Switzerland)

In the context of a new project in structured document recognition, we address the problem of designing a software architecture which is able to integrate all the necessary, but heterogeneous knowledge. Starting from the needs of the *CIDRE* project, we propose a concrete framework built upon existing software pieces, and follow a multi-agent paradigm. DAFS serves as the main document management package. The computational engine is written on a distributed and multi-threaded platform, and an original coupling with a GUI is presented. To demonstrate the validity of the approach, a prototype that interactively recognizes the entire physical structure of an input document has been developed.

1 Introduction

The need for structured documents has been recognized for a long time and their importance increases with the development of new software applications. The field of document structure recognition has reached a point where we must admit that, except for extremely confined applications, a fully automatic system that always leads to valuable results is an illusion. More specifically, the systems developed so far showed that:

- recognition results are subject to errors that have to be corrected manually; because of the uncertainty cumulated at each step of analysis, no algorithm will achieve a negligible error rate;
- the definition of the logical structure over a document class depends on the application needs; moreover, describing formally a document model is an extremely tedious task;
- structure models have often to be tuned to accept the particularities of a specific document, mainly because practical samples were not produced in a rigorous way, for example with a typewriter or a with weak-structured editor like MS-Word.

We believe that only interactive tools can overcome these deficiencies and enhance the productivity of document re-encoding to a satisfactory level. The

CIDRE[a] project[3] comes from a general re-evaluation of structured document recognition, with an emphasis on three main topics:

Assistance from the human operator: our ambition is not to develop an automatic system, but rather an assisted environment; this brings new constraints in the design of the automatic functionalities.

Automatic inference of document models: as a manifestation of a true human-machine cooperation, the document models must be incrementally learned from examples during the recognition sessions.

Cooperative architecture: we assume that the first two objectives won't be reached without revising the software environment. To approximate inside the code the idea of collaboration, we determined that the best approach was a multi-agent design.

This paper focuses on the design of a software architecture that can support the programming challenges of *CIDRE*. This is in itself an important research issue, because in document image analysis as in many other domains, it is a real challenge to integrate coherently all the existing knowledge; that's why the effort should not only be put on inventing new algorithms for specific subtasks, but also on finding a relevant framework to combine various software components. Section 2 addresses the architectural problems raised by the specification of the desired system. Our proposition is presented in section 3, in terms of the integration of existing software components. We show then in section 4 how the proposed framework was successfully used for a restricted application, and what kind of problems are expected in a generalization of the prototype.

2 Architecture Requirements for the *CIDRE* Project

The *CIDRE* project is quite ambitious, and its realization imposes strong requirements on the underlying software architecture. We emphasize in this section the design problems raised by the human-machine cooperation, the knowledge maintenance, the solution search facilities and the physical distribution of time-consuming tasks.

2.1 High Interactivity

The starting point of the project consists in revising the functionalities of the recognition system: we drop the quest for a full automatic recognition

[a] For Cooperative & Interactive Document Reverse Engineering.

tool, aiming instead at the development of an environment that offers help for document restructuring; in other words, we are facing a special case of computer aided data interpretation [13].

Instead of joining manual correction facilities with a tool that excludes them by nature, the whole system design has to include the idea of assisting the user. We believe that defining a convenient environment is not merely a question of graphical presentation, because placing the human operator as the real master of the recognition task has direct consequences on the system architecture. This means that user interventions should influence the session engine. For example, the manual correction of a segmentation result would be noticed to adapt some threshold of the responsible algorithm. Actually, most system components have to support a connection with the end-user. That's why we propose a general mechanism to couple the application core and the GUI, which is described in subsection 3.4.

2.2 Knowledge Integration

The project encompasses three programming issues:

Data: the documents being processed are rich collections of structured information. We distinguish first the specific document (e.g. a precise article scanned from DAS proceedings) and the generic model (e.g. the class of DAS proceedings articles). While it is clear that the document instance evolves during the recognition session, it is a *CIDRE* originality not to consider the model as a passive configuration parameter, but as incremental knowledge. We want the recognition system to learn the models from examples, or to reuse (parts of) existing generic descriptions. The second partitioning concerns physical (layout or formatting result) and logical (reflecting the intention of the writer) structures [12].

Tasks: the system has to show various computing skills, namely feature extraction, interpretation expertise, logical labeling and model inference. These four topics are broad and very complex. Most research is done on precise subproblems; we are more interested in the integration of knowledge, which is a promising but far from trivial orientation.

Control levels: the different parts of the recognition environment cannot be written easily with the same programming abstractions: you are not in the same background when you deal with an RLSA algorithm, a search strategy or with widgets. Figure 1 shows these control levels. The horizontal axis classifies functionalities from automatic computing to manual

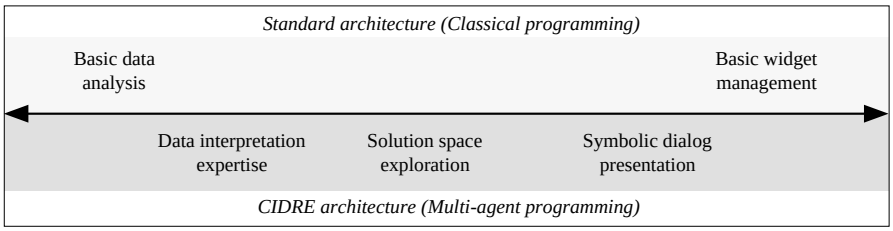

Figure 1: Control levels in *CIDRE*.

interventions. Both extremities fall into well-understood programming styles. But they will be driven by higher-level managers in order to serve at best the final goal of solution search.

Globally the integration problem is a question of session modeling: we have to identify and specify relevant parts in the system, defining clearly the role of each one and the relations between them.

2.3 Concurrent Exploration of the Solution Space

A major characteristic of our application is the omnipresent uncertainty. From the initial inputs (noisy images) to the results of analysis tools (OCR, segmentation, logical labeling), nearly no information can be totally trusted. It is very important that each component is aware of working only on hypotheses, not on established facts. For example, the ASCII interpretation of a word may vary according to the applied OCR method or to parameters of the call. As a corollary, the system must offer full support for hypotheses generation, solution extension or critics, backtracking, reinterpretation and so on. Besides, it is a privileged situation for automatic analyzers to try collaboration and negotiation protocols (e.g. between a font recognizer and a mono-font OCR [22]). We are studying new approaches of document modeling, learning and logical recognition [7]. After a lot of discussion about solution search schema, we are now convinced that:

- local alternatives should not be abandoned until high confidence in one of them is established;
- the solution space should be explored concurrently; the user should not be constrained to follow a rigid evolution scheme imposed by the system;

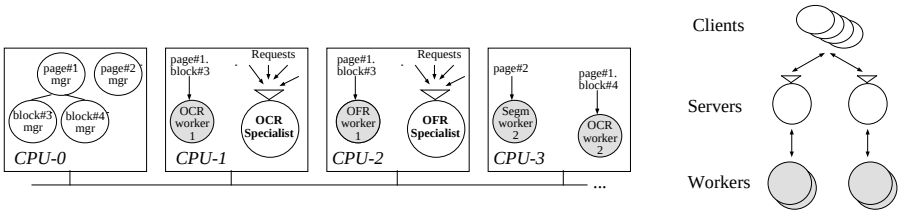

Figure 2: Task distribution over the network with client-server-worker schema.

- the system should be designed to make intensive use of the paradigm of *local revision*.

2.4 Physical Parallelization of Heavy Processing

Document image analysis comes with its unavoidable panoply of time-consuming algorithms. However, long delays are not easily tolerated in a convenient environment. One way to improve the performance of the system is to rely on parallel programming. Anyway, the general trend in hardware evolution is to build multi-processor machines. We want to express the inherent parallelism of our application, in such a way that the program can exploit the available computing power, be it distributed over a network of workstations or inside a supercomputer. Figure 2 shows a system that can take advantage of the independence between data and also between tasks. For example, distributed processing can be tuned to perform simultaneously OCR and Optical Font Recognition (OFR) on the same block, or to segment two distinct pages. The paradigm of client-server-workers seems useful in this context; for instance we could have a dedicated OFR server that can master several workers to execute the requests submitted by any part of the document processor.

3 Salient Features of the Proposed Framework

Setting up a software framework to hold *CIDRE* means looking for existing packages in various domains. This section presents our concrete proposition. First we show how the system is decomposed with a multi-agent design. Then the document management and processing tools are summarized. The third subsection is devoted to an implementation platform that deals with parallelism and concurrency. Finally we present an original mechanism to assemble the computational engine of an application and the dialog engine of its user interface.

3.1 Multi-agent Modeling

Document recognition falls into the category of complex problems, in the sense that its resolution in its generality does not follow a strict deterministic algorithm, but rather requires real expertise based on a large variety of (maybe automated) data analysis. To approximate this scheme, very interesting results have been achieved using fuzzy rules [11] or blackboard systems [8]. For our part, we are sure that Distributed Artificial Intelligence (DAI) techniques will bring a decisive contribution, so we adopt an approach that focuses on control decentralization, local reasoning, collaborative behavior, living partial results, and human supervision.

Talking about multi-agent modeling is a bit risky, because there is no consensus yet on the definition of DAI, Multi-Agent Systems or Distributed Problem Solving [10]. Instead of taking a position in that lingering debate, we simply want to pick up both the general spirit and concrete techniques, then have practical experiences with their application, hoping that an intuitive rather than formal notion of agent is not too big an obstacle.

Our multi-agent society is shown in Figure 3. It is first composed of a set of partial solution managers, that respect the natural organization inside a recognized document. The specific document is fragmented into physical (pages, blocks, lines, etc.) and logical (parts, fragments, strings, etc.) tree structures. The same distinction applies to the generic model, where hierarchy denotes specialization (not composition) relations, issued from the predefined entities. These *data-agents* can submit requests to so-called *specialist-agents* representing a data-interpretation expertise, such as OCR or segmentation. Finally, each member of this society has its counterpart GUI agent, responsible for the dialog with the user (see subsection 3.4). The behavior of all these agents during a recognition session is discussed in section 4.

3.2 Document Handling Framework

One of the first decisions to start the realization concerned data integration: there must be a clear organization of all the information used by the system. Our proposition, presented in [4], stands on the following bases:

- we try to support all possible kinds of relations between components of a document (specific and generic, physical and logical); it is important that the session modeling respects some sound theoretical concepts about document structures.

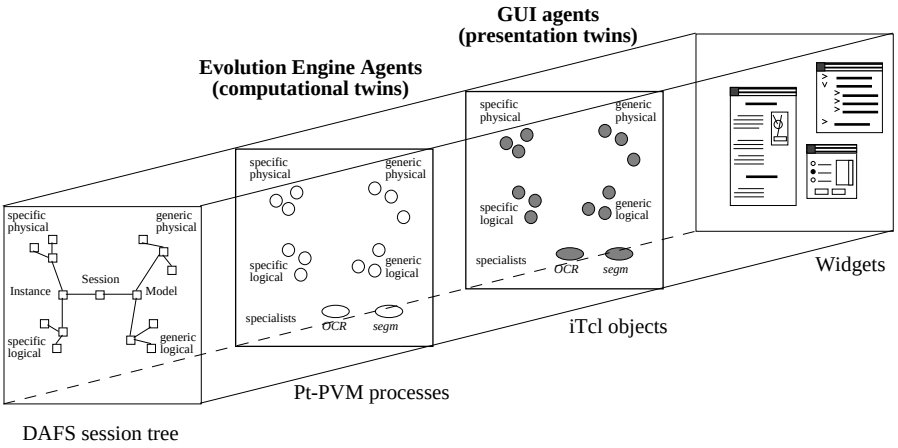

Figure 3: Multi-agent society in *CIDRE*.

- the whole recognition session is represented with a DAFS structure. DAFS is a package [b] composed of a format definition for document image re-encoding, an API and an editor [19]. We defined a specialization of DAFS to represent a complete session in a consistent way, via a full set of new entity types with their associated properties, and the encoding of a recognition session inside one structure.

Together with the multi-agent model, we can now specify the paradigm of active data: as suggested in Figure 3, we associate a partial solution manager process with each node of the DAFS hierarchical structure representing the current session results. Once the data representation was chosen, we integrated a collection of automatic analysis tools in C, including (for the moment):

- an image processing library, with special routines for document preprocessing, such as skew detection, rotation or filtering;
- a complete segmentation package [2] that can recognize the physical micro- (from signs to blocks) and macro-structures (from blocks to page), as well as tables, formulas, graphics or photos;
- a set of text recognition programs, composed of the commercial OCR system ScanWorX [17] which also gives segmentation results, a home-made

[b] Available from the DIMUND server at `http://documents.cfar.umd.edu/`.

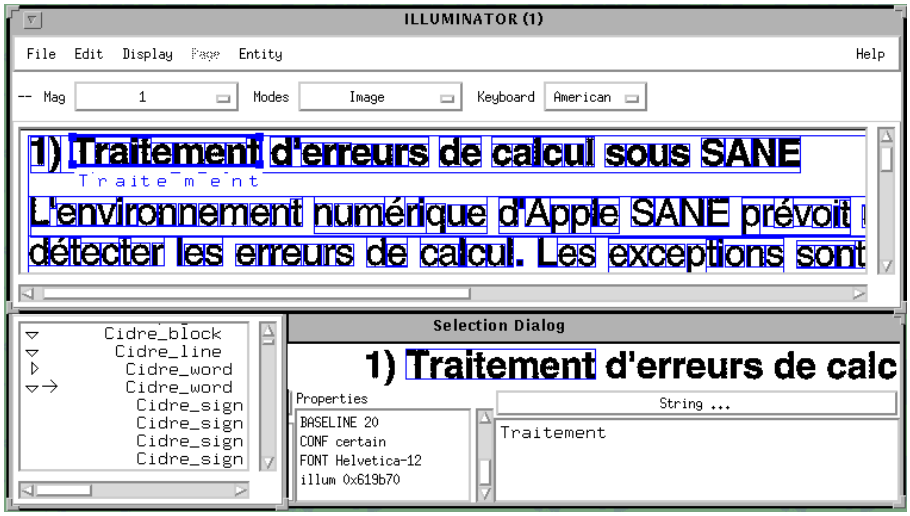

Figure 4: Page image re-encoded with our DAFS tools.

mono-font OCR, and the *ApOFIS* OFR system that detects fonts directly from images [22];

- a two-fold PostScript converter[c] that builds page nodes with images and sign nodes with position and ASCII code (using respectively *ps2tiff* and *ps2ascii*, two utilities distributed in the *GhostScript* package);

Figure 4 shows the DAFS structure currently generated from re-encoding document images, as presented by the DAFS editor `Illuminator`. We can see the hierarchical physical structure (boxes on the image and tree summary), augmented with ASCII interpretation and with typographic properties.

As another document management platform, we plan to convert our document samples or models from/to Thot [18], because it is a powerful structured document production system. Its main advantages are a sound theoretical background, a full API, an excellent editor, converters for formats like LaTeX or SGML, and a wide community of researchers using in the product.

[c] Re-structuring of electronic documents is also an application for *CIDRE*; the problem is not far from recognizing scanned pages. Simply, some information can be extracted in a better way (noise-free images, doubtless font or ASCII codes).

3.3 Distributed and Multi-threaded Platform

The choice of a programming platform that offers relevant concurrent and parallel computation facilities is quite hard. Hopefully, we can take advantage of the studies of the parallelism research group of our institute. One of its challenges is the development of a coordination language for massively parallel systems [1]. In this context, they realized the Pt-PVM system [14], with the following characteristics:

- it consists mainly of a C/C++ library offering primitives to declare and spawn processes of different granularity, and to perform message passing between them;
- it is built upon PVM, the most popular message-passing platform for heterogeneous networks, where the computation space is a set of connected workstations; the current realization of Pt-PVM runs on Solaris 2.4 and is being ported to a Parsytech parallel machine (the PowerXPlorer, under Parix operating system);
- every aspect of the coordination is carefully addressed, leading to a clearly defined concept of correspondent;
- preemptive threads [20] are fully supported. Inside the same heavyweight process, these lightweight processes share a common address space;
- a very clean solution is proposed to allow collaboration with other Unix applications, connected through pipes or sockets (see next subsection);

This platform is particularly suited for the *CIDRE* project, as it can offer both concurrency and distribution, in the same programming environment as the document handling framework.

3.4 Coupling Evolution Engine and GUI

Human-machine dialog is a major part of the system, raising the problem of the integration of a GUI with global session management. We contend that GUI programming is still an open question in software development. In spite of well-defined conceptual models (like SmallTalk's MVC or PAC [9]) as well as standard look-and-feel toolkits (like Motif), the programmer has often to cope with practical constraints that lead to unpleasant encoding. In fact, GUI programming seems intrinsically tedious, whatever language is used (be it object-oriented, script-like, declarative, etc.). It is not easy to mix event-based programming with the other control paradigms used to write the rest of

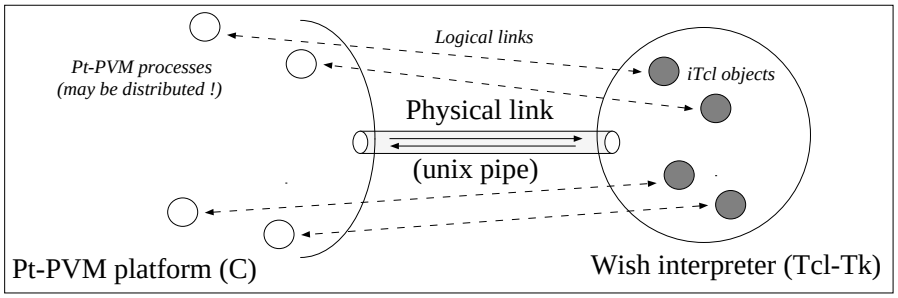

Figure 5: C threads and GUI objects – Logical and physical links.

the application. Moreover, an ideal GUI has to use concurrency[d], because the dialog should never appear to be blocked. Finally, there is always the problem of finding the best frontier for dialog components: even if interface builders are helpful for setting up presentation details, there remains the "grey zone" (e.g. callbacks) that connects the GUI to the application kernel. To overcome some of these difficulties, we propose the following mechanism:

- The GUI is written in Tcl-Tk[16] (and their object-oriented extensions iTcl and iTk[e]), and will run in a "wish" interpreter as a separate Unix process. C code inside Pt-PVM realizes the computational part and iTcl code the dialog part of the application. This linguistic distinction prevents the programmer from writing code that mixes GUI management with algorithmic flow. It is a frequent pitfall to include widget management inside a module in a prototype.

- To each C thread we can associate an iTcl object, thus defining the concept of *twin agents*. Each thread corresponds to an iTcl class. Two twins are logically connected by a direct communication link, although physical transfer is implemented indirectly through a unique Unix pipe, as illustrated in Figure 5.

- Inside the twin couple, roles are natural: the C thread controls the computational part of the agent, and the iTcl object formats it to be viewed and controlled by the user, through the management of widgets (see Figure 3). The message passing protocol between both twins models the

[d] And it is not so easy as the X11 library is not thread-safe.
[e] See e.g. `http://www.wn.com/iwidgets`.

Partial Solution Manager (C excerpt)	GUI Agent (iTcl excerpt)

`void page_thread (THREAD_ENV *p) {`
`/*--- initialization behavior ---*/`
`spawn_itcl_twin("Page_twin", p);`
`do { cidre_receive_msg(...);`
`/*--- reactive behavior ---*/`
`switch (msg_kind) {`
`case addBlock: /* eg from segmenter */`
`...`
`send_to_twin(p,"show_rect",coord);`
`case applyOcr: /* eg from iTcl twin */`
`/*--- perform OCR with <msg_tail>`
`as confidence threshold ---*/`
`} while (1); }`
	`itcl_class Page_twin { inherit Data_twin`
	`#--- initialization behavior ---`
	`method constructor {...} {`
	`#--- create widgets, show image ---`
	`}`
	`#--- reactive behavior ---`
	`method show_rect {x y w h} {`
	`#--- draw a rectangle ---`
	`}`
	`method apply_ocr_menu_cmd {} {`
	`#--- ask the user for a threshold ---`
	`send_to_twin "$apply_ocr_MSG $thresh"`
	`} }`

Figure 6: Expressiveness of the relations between twins.

presentation-independent interactive functionalities. This scheme permits then to develop and test both parts independently.

- Inside each world, agents may not be organized the same way or according to the same topology. This means that two agents may have direct interactions only for presentation purposes (e.g., a word and a page), or only for computational reasons (e.g., a block and the OCR specialist).

- Although we are mixing two rather different programming styles, the solution we propose makes it possible to express the relations between couples in an elegant way. Figure 6 is an excerpt of agent code showing the coordination primitives.

Despite not being a revolutionary method of GUI programming, this proposal is nevertheless a concrete step towards writing smart code to couple GUI and C applications. More generally, the twin paradigm could be used to couple any programming environment [5], with a control abstraction that simulates agents (objects, communicating processes, logical rules). In *CIDRE* for instance, where document handling is in C and the GUI is in iTcl, we could use the constraints programming facilities of Oz [21] to perform the solution search itself, adding Oz objects as a third kind of twin.

4 Evolution Strategies

The static aspects of the multi-agent society were presented in subsection 3.1. Once the agent kinds are chosen, it is still difficult to specify their roles in determining the system behavior. These dynamic aspects are addressed in this section. First the realized prototype and its different participants are

described, and a scenario is discussed. Then we sketch some expected problems on the way towards a complete environment, leading to the conclusion that balancing the competences is a crucial goal.

4.1 The Realized Prototype: a Full Physical Structure Generator

Obviously the recognition environment has to be realized step by step. So we started the development of the software architecture for a restricted application, namely the assisted recognition of the entire physical structure, i.e. the tree of physical entities from pages to signs, with font and ASCII interpretation of the text encoded in attributes on the DAFS entities. The goal is to test the validity of our approach, not to optimize performance, quality of automatic analyzers or ergonomics. Figure 7 sketches what is being displayed at runtime, i.e. the session window showing the current page structure and some dialog elements, two tracers (for Tcl and C), and the Pt-PVM supervisor. We will now describe the behavior of each kind of agent running in this testbed program.

Physical entities: the active data managers are restricted to specific physical entities. The volume agent maintains the permanent session root; it can perform I/O operations such as append a new page or save/load a session. The page, block and line agents maintain the coherence in their decomposition. They support various messages to modify their different attributes, or their descendents. At any time the specialists can be invoked. Words and signs are directly managed by the line agents.

Specialists: three kinds of specialists have been implemented. The segmenter and OCR accept separate requests to analyze the whole page image or a precise sub-entity. Many parameters can be tuned, especially for ScanWorX. The font recognizer is able to compute a ranked list of possible fonts, among a selected font set; it can also be invoked to estimate a confidence rate for a font proposed by another agent (maybe the user or in the future a logical labeler).

GUI twins: the GUI volume agent maintains permanent widgets such as the menu bar, whose callbacks activate the volume thread. The page twin creates a Tk canvas that will hold the image, all sub-entity envelopes, and the recognized text. Pop-up menus are attached to each displayed item, such that the events are redirected to the agent managing it; this lets the user conduct a dialog with the agent population, e.g. in order to correct the segmentation, OCR or OFR results. Finally, the specialist twins present some control panels to tune the automatic analyzers.

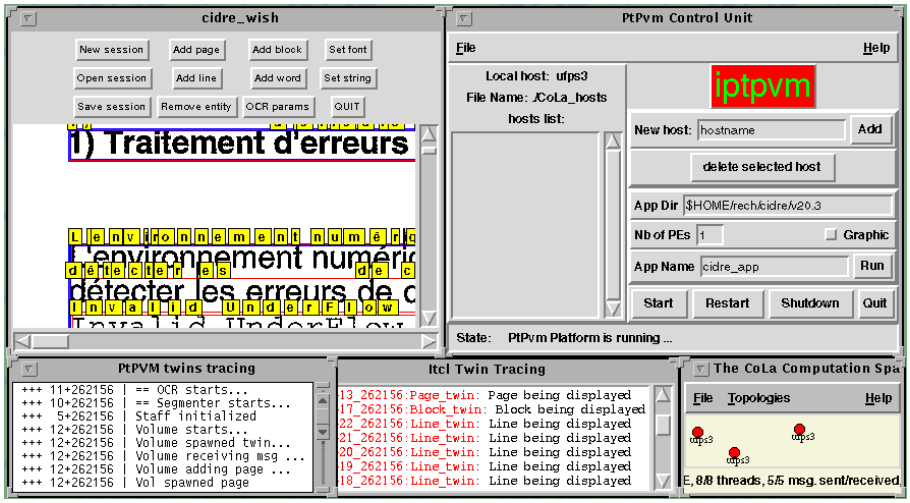

Figure 7: Our prototyping testbed.

4.2 A Typical Scenario

Figure 8 sketches the evolution of the system during the analysis of a new page. Every agent behavior is represented by a vertical thread showing its state transitions. Interactions are either agent spawning or message passing.

When a new page is created, the corresponding data agent generates its presentation twin, and submits a request to the segmenter. This specialist finds one of its remote workers to perform image analysis; when the job is over, the segmenter translates the results to a sequence of `add-child` messages. This re-activates the computational page twin which now merges these new block sub-trees with its existing children (if any). If no match is found, the new entity is accepted and a block agent is spawned. This solicits the OCR in an analogous way (the OCR worker is not represented in the figure).

The interesting point is that `add-child` messages can be issued from different sources, notably from the user (via the presentational twin). In general the user is free to give his own interpretation of the data at any time, e.g. to set the font of the block before the analyzer is called on every line. When a new result causes a conflict (here the block faces two incompatible line decomposition), the data agent can invoke a revision of the former interpretation, or catch the operator's attention through the GUI.

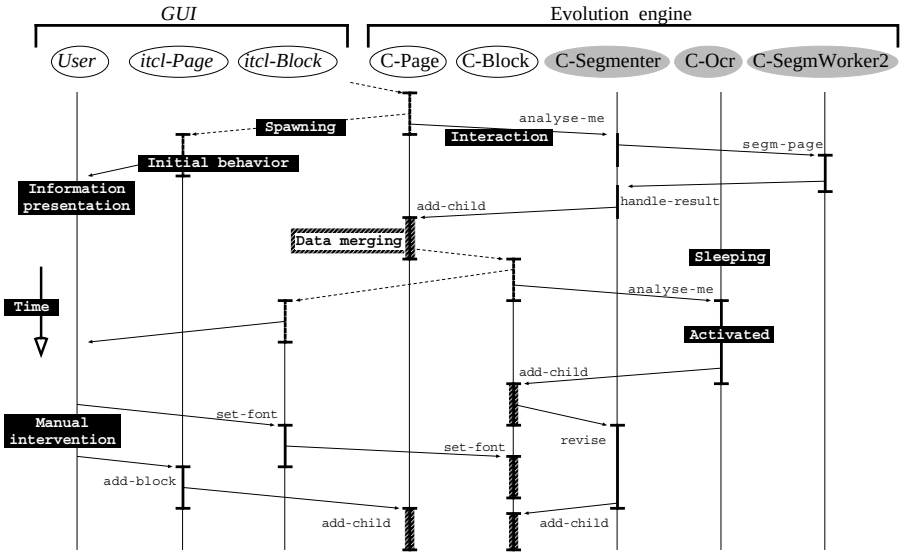

Figure 8: Evolution of a typical scenario.

4.3 Towards a Complete Environment

The prototype is greatly simplified with respect to the desired re-structuring environment. The specific document will be extended with logical labels, bringing composition relations between both structures. But the real gap is to integrate the document model, and to program an intelligent method for solution space exploration.

Generic entities: like the document instance, the model is fragmented into cooperating agents. The generic physical entities are responsible to cluster the instances coherently (e.g. normal, bold, and tiny lines); these clusters are of central importance as they condition the logical recognition. The generic logical entities have to maintain both syntax and presentation rules, which are used by the logical parsing-matching algorithm. Managing relations between classes and their instances (both evolving during the session) is quite complex.

Intelligent evolution: in the tested prototype, the computation schema is systematic. But as we stated in subsection 2.3, we want to design a flexible recognition method, that asks for local revision when a portion of the solution

space seems too weak. Ultimately, it is the *cooperation* phenomenon that is the objective. For instance, the evolution engine agents will support a confidence measure to invoke deeper expertise (maybe from the user) when needed, i.e. in the presence of unstable hypotheses. This will imply firstly to study the general problem of data merging, secondly to develop concrete algorithms such as a negotiation protocol. It is never easy to encode this kind of intelligent system behavior.

4.4 Finding a Harmonious Balance of Competences

The great challenge of the integration quest is to reach the most elegant expression of data/control flows. From our practical experience, there are many different ways of arranging the competences inside a multi-agent society. For example, you have to decide whether it is the active data managers or the specialists that perform the merging of a new result with the current solution. A similar case arises if we want to adapt the OCR method according to properties of the image area: who will be concerned by this refinement, the block agents, the OCR specialist, or a generic block agent responsible to classify the samples ?

The key point is to balance the competences so that each participant is complex enough to offer the advantages of intelligence distribution, but simple enough to play a well-defined role with only local capabilities. In multi-threaded programming, this problem of task delegation is raised again at the implementation level, when we must decide whether information is accessed through message passing or through shared memory.

5 Conclusion

This paper discussed many aspects of a new approach to deal with document image analysis problems [15], especially those related to the software design of the system. The major contribution is a real effort in know-how integration. This led to the development of an integrated testbed designed as a multi-agent society, which is used to evaluate different evolution strategies. The accent is put on the combination of existing technologies:

- standard support for document image re-encoding with DAFS;
- structured document editing with Thot;
- automatic analyzers (segmenter, font detector, OCR, PostScript tools);
- decentralized control with multi-agent design;
- distributed and multi-threaded programming with Pt-PVM;

- original GUI coupling between C and Tcl.

One of the main drawbacks in our proposition is the present lack of a high-level reasoning language that can express "intelligent" behavior; we do not abandon the idea, but it seems that this technology has not yet reached a sufficient stability. On the other side, the promising experiences encourage us to study the modeling of session evolution [6], and in parallel to extend the functionality of the basic prototype. Globally, our architecture sketches some possible software bases for the assisted recognition of scanned pages, but also for any transformation of electronic documents into a well-structured form.

Acknowledgments

The *CIDRE* project is supported by the Swiss National Fund for Scientific Research, code 21-42'355.95.

References

1. M. Aguilar, B. Hirsbrunner, and O. Krone. CoLa : a coordination language for massively parallel systems. In *Proceedings ACM Symposium on principles of Distributed Computing (PODC)*, Los Angeles, California, August 1994.
2. A. Azokly. *Une approche générique pour la reconnaissance de la structure physique de documents composites.* PhD thesis, IIUF-Université de Fribourg, 1995. n. 1105.
3. F. Bapst, R. Brugger, and R. Ingold. Towards an interactive document recognition system. Internal working paper 95-09, IIUF-Université de Fribourg, March 1995.
4. F. Bapst, R. Brugger, A. Zramdini, and R. Ingold. L'intégration des données dans un système de reconnaissance de documents assistée. In *CNED'96*, Nantes (France), 1996.
5. F. Bapst and O. Krone. A coordination kernel for coupling heterogenous programming environments. Internal working paper 97-03, IIUF-Université de Fribourg, February 1997. Submitted to COORD'97.
6. F. Bapst, A. Zramdini, and R. Ingold. A scenario model advocating user-driven adaptive document recognition systems. Technical report, IIUF-Université de Fribourg, 1996. To appear in ICDAR'97.
7. R. Brugger, A. Zramdini, and R. Ingold. Modeling documents for structure recognition using generalized n-grams. Technical report, IIUF-Université de Fribourg, 1996. To appear in ICDAR'97.

8. Y. Chenevoy. *Reconnaissance structurelle de documents imprimés: études et réalisations.* PhD thesis, CRIN-Nancy, Décembre 1993.
9. J. Coutaz. *Interfaces homme-ordinateur.* Dunod informatique, 1990.
10. Y. Demazeau and J.-P. Müller, editors. *Decentralized artificial intelligence*, volume European Workshop on Modelling Autonomous Agents in a Multi-Agent World. North Holland, 1990.
11. T. Hu. *New methods for robust and efficient recognition of the logical structures in documents.* PhD thesis, IIUF-Université de Fribourg, 1994. n. 1076.
12. R. Ingold. *Une nouvelle approche de la lecture optique intégrant la reconnaissance des structures de documents.* PhD thesis, EPFL, Lausanne, 1988. n. 777.
13. E. Kant. Interactive problem solving using task configuration and control. *IEEE Expert*, 3(4):36–49, 1988.
14. O. Krone, M. Aguilar, and B. Hirsbrunner. Pt-PVM : Using PVM in a multi-threaded environment. In *Euro-PVM*, Lyon (France), September 1995.
15. G. Nagy. Document image analysis: what is missing? In *Image analysis and processing (ICIAP'95)*, volume 974 of *Lecture notes in computer science*, pages 577–587. Springer, San Remo (Italy), 1995.
16. J. K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley Publishing Company, Inc., 1994. available on ftp sites.
17. B. Paddock and T. J. Platt. *ScanWorX API, Programmer's Guide.* Xerox Imaging Systems, Inc., 9 Centennial Drive, Peabody, Massachusetts 01960, 1992.
18. V. Quint, H. Richy, C. Roisin, and I. Vatton. *Thot – Manuel utilisateur.* Imag - INRIA, November 1995. V0.95, previously called Grif.
19. RAF Technology, Inc. *DAFS library, programmer's guide and reference*, August 1995.
20. D. E. Ruddock and B. Dasrathy. Multithreading programs: Guidelines for DCE applications. *IEEE Software*, 13(1), January 1996.
21. G. Smolka. An Oz primer. Technical report, German Research Center for Artificial Intelligence (DFKI), January 1995. Draft version.
22. A. Zramdini. *Study of optical font recognition based on global typographical features.* PhD thesis, IIUF-Université de Fribourg, 1995. n. 1106.