Institut d'Informatique de l'Université de Fribourg (Suisse)

Reconnaissance de documents assistée : architecture logicielle et intégration de savoir-faire

Thèse de doctorat

soumise à la Faculté des Sciences de l'Université de Fribourg (Suisse) pour l'obtention du grade de Doctor Scientiarum Informaticarum

Frédéric BAPST

de La Roche (FR)

Thèse n° 1228

Imprimerie St Paul, Fribourg

1998

Acceptée par la Faculté des Sciences de l'Université de Fribourg, sur la proposition des Professeurs :

- Rolf Ingold, Université de Fribourg, Suisse;
- George Nagy, Rensselaer Polytechnic Institute, Troy, New York, USA;
- Georges Stamon, Université de Paris V, France.

Fribourg, 2 octobre 1998

Le Directeur de thèse :

Prof. Rolf Ingold

Le Doyen :

Prof. Béat Hirsbrunner

Cette thèse est disponible sur WWW à l'adresse `http://www-iiuf.unifr.ch/~bapst`

Remerciements

Arrivé au terme de ce doctorat, je tiens à exprimer mes remerciements envers toutes les personnes qui m'ont aidé de près ou de loin durant ces cinq ans d'études. Certaines contributions méritent d'être relevées tout spécialement :

- Rolf Ingold m'a offert un encadrement idéal, et m'a dirigé de manière fort judicieuse. Il a parfaitement su m'initier au monde scientifique, en me transmettant à la fois une formation des plus solides dans son domaine, une rigueur dans la démarche du chercheur, et le plaisir de s'ouvrir à d'autres disciplines.
- George Nagy m'a toujours impressionné par la fraîcheur de ses idées. Ses remarques détaillées et pertinentes sur la première version de ma thèse m'ont aidé à l'améliorer considérablement. Je lui suis très reconnaissant d'avoir participé à ma défense en tant qu'expert. Ce fut un honneur pour nous de recevoir l'appui d'un spécialiste si prestigieux en reconnaissance de documents.
- Georges Stamon a depuis longtemps manifesté de l'intérêt pour mes travaux, jusqu'à accepter de joindre le jury de thèse. Venant d'une personnalité si compétente et sympathique, ses commentaires m'ont beaucoup touché.
- Lyse Robadey a été la première, puis la dernière relectrice de ce mémoire, dont elle a permis d'augmenter sensiblement la clarté. Davantage encore, je lui dois une grande part de ma motivation.
- Oliver Hitz s'est chargé de réaliser l'OCR décrit dans la section 8.2. Il a eu le grand mérite d'amener son projet jusqu'à un produit opérationnel.
- Oliver Krone m'a apporté une assistance indispensable dans l'utilisation, voire l'adaptation de PT-PVM, dont j'étais le premier «beta-tester».
- Mes collègues m'ont offert une ambiance de travail inoubliable. Merci donc à Rolf Brugger, Folco Banfi, Houda Chabbi, Abdu Zramdini, Lukas Theiler, et bien d'autres.
- Ma nièce Aurélie a accepté d'illustrer chaque chapitre par un dessin plein d'imagination et de malice. Chapeau !
- Sylvie n'a jamais douté de mes capacités. Son tendre soutien m'accompagnait en permanence.
- Je remercie encore ma famille, mes potes de Villaz, Zina, Cyrano, Salomé, les heures calmes, les poursuites...

A mes parents,
Très tendrement...

Table des matières

1 Introduction **1**

1.1 Problématique de la reconnaissance de documents . . . 1

1.1.1 Généralités sur la gestion de documents . . . 2

1.1.2 Analyse d'images de documents . . . 3

1.2 Avenir des systèmes de reconnaissance . . . 4

1.2.1 Vers de nouveaux besoins . . . 4

1.2.2 Vers une démocratisation de la technologie . . . 5

1.2.3 Vers une reconnaissance assistée . . . 6

1.3 Objectifs de ce travail . . . 7

1.3.1 Conséquences d'une approche assistée . . . 7

1.3.2 Architecture logicielle pour la reconnaissance assistée . . . 8

1.3.3 Intégration de savoir-faire . . . 9

1.4 Organisation en chapitres . . . 9

2 De la reconnaissance automatique à la reconnaissance assistée **11**

2.1 Survol de l'état de l'art . . . 11

2.1.1 Etapes du processus de reconnaissance . . . 12

2.1.2 Problèmes pratiquement résolus . . . 12

2.1.3 Ce qu'on essaie de reconnaître . . . 13

2.1.4 Quelques prototypes complets . . . 14

2.1.5 Quelques projets proches de *CIDRE* . . . 15

2.2 Limites des systèmes actuels . . . 16

2.2.1 Rôle de l'utilisateur . . . 16

2.2.2 Manque de souplesse . . . 17

2.3 Révision de la problématique dans le projet *CIDRE* . . . 18

2.3.1 Reconnaissance assistée par ordinateur . . . 18

2.3.2 Réingénierie de documents au sens large . . . 19

2.3.3 Rôle de l'architecture logicielle . . . 19

2.3.4 Modèles de documents . . . 19

2.4 Amorces pour la conception d'un système assisté . . . 20

2.4.1 Ergonomie ciblée sur une application . . . 20

2.4.2 Panoplie d'analyseurs . . . 20

2.4.3 Allocation des ressources . . . 21

2.4.4 Métaphores . . . 22

2.5 Champs d'application pour *CIDRE* . . . 23

2.5.1 Réédition de documents scannés . . . 23

2.5.2 Transformations de documents électroniques . . . 23
2.5.3 Archivage et indexation . . . 24
2.5.4 Synthèse vocale de documents structurés . . . 25

3 Dialogue homme-machine pour la reconnaissance de documents 27
3.1 Eléments théoriques en interaction homme-machine . . . 28
3.1.1 Survol du domaine . . . 28
3.1.2 Modélisation des interactions . . . 29
3.1.3 Apports principaux au projet *CIDRE* . . . 30
3.2 Définition informelle de la coopération homme-machine . . . 31
3.2.1 Interactivité . . . 31
3.2.2 Adaptabilité . . . 32
3.2.3 Coopération homme-machine dans *CIDRE* . . . 32
3.3 Rôles de l'homme et de la machine dans un système de reconnaissance . . . 33
3.3.1 Survol des fonctionnalités . . . 33
3.3.2 Stratégie d'analyse . . . 34
3.3.3 Scénario de reconnaissance . . . 35
3.4 Styles de dialogue adaptés à *CIDRE* . . . 36
3.4.1 Environnement d'invalidation . . . 36
3.4.2 Enregistrement de macros-fonctions . . . 37
3.4.3 Collaboration faiblement couplée . . . 37
3.5 Interactions dans les sous-tâches de reconnaissance . . . 38
3.5.1 Segmentation assistée . . . 39
3.5.2 Reconnaissance de fontes assistée . . . 39
3.5.3 OCR assisté . . . 41
3.5.4 Etiquetage logique assisté . . . 41
3.6 Incidences sur l'architecture logicielle . . . 42
3.6.1 Architecture centrée sur l'utilisateur . . . 42
3.6.2 Charpente réutilisable . . . 43

4 Gestion des données 45
4.1 Inventaire des informations à intégrer . . . 45
4.1.1 Description de documents . . . 46
4.1.2 Etat de l'interface homme-machine . . . 47
4.1.3 Configuration des analyseurs . . . 47
4.1.4 Progression dans l'espace d'analyse . . . 47
4.1.5 Entrées-sorties . . . 48
4.2 Organisation des données . . . 48
4.2.1 Entités dominantes et attributs . . . 49
4.2.2 Relations dominantes . . . 50
4.3 Structures de données . . . 52
4.3.1 Représentation uniforme . . . 52
4.3.2 Paquetage DAFS . . . 52

4.3.3 Spécification basée sur DAFS . . . 54

5 Conception en système multi-agents 57

5.1 Apport de l'intelligence artificielle distribuée . . . 58
5.1.1 Principe de localité . . . 58
5.1.2 Non déterminisme . . . 59
5.1.3 Aspects de génie logiciel . . . 60
5.2 Dichotomie figuratif/opératoire . . . 60
5.2.1 Agents-données . . . 60
5.2.2 Agents spécialistes . . . 61
5.3 Dichotomie automatique/interactif . . . 62
5.3.1 Agents d'analyse . . . 62
5.3.2 Agents de dialogue . . . 63
5.4 Fonctionnement du système . . . 63
5.4.1 Topologie . . . 64
5.4.2 Interactions . . . 65
5.4.3 Règles de comportement . . . 65

6 Plateforme d'implémentation 69

6.1 Support de programmation requis . . . 69
6.1.1 Exécution asynchrone . . . 70
6.1.2 Exploitation du parallélisme . . . 70
6.1.3 Extensibilité . . . 71
6.1.4 Rapide prototypage . . . 72
6.2 Programmation concurrente et distribuée avec PT-PVM . . . 72
6.2.1 PT-PVM . . . 72
6.2.2 Processus légers et mémoire partagée avec PT-PVM . . . 74
6.2.3 Processus lourds et configuration en réseau . . . 75
6.3 Programmation multi-langages avec C et Tcl-Tk . . . 76
6.3.1 Approche multi-langages . . . 76
6.3.2 Interface graphique basée sur Tcl-Tk . . . 77
6.3.3 Couplage expressif . . . 79
6.4 Aperçu du prototype réalisé . . . 80
6.4.1 Caractéristiques techniques de notre prototype . . . 80
6.4.2 Exemple de scénario . . . 81

7 Méthodes d'analyse 85

7.1 Critères de qualité . . . 86
7.1.1 Simplicité . . . 86
7.1.2 Paramétrisation . . . 86
7.1.3 Apprentissage incrémental . . . 87
7.1.4 Compatibilité . . . 88
7.2 Interdépendances entre analyseurs . . . 89

7.2.1 Compétition . . . 89
7.2.2 Producteurs-consommateurs . . . 89
7.2.3 Protocoles de coopération . . . 89
7.2.4 Variété des chemins d'analyse . . . 90
7.3 Exemple d'intégration en typographie . . . 91
7.3.1 Notion de fonte en reconnaissance de documents . . . 91
7.3.2 Support logiciel pour la gestion des fontes . . . 92
7.3.3 Apport des connaissances typographiques . . . 94

8 Réalisations et expériences 99
8.1 Démarche expérimentale . . . 99
8.1.1 Objectifs . . . 99
8.1.2 Base de données pour les tests . . . 100
8.2 OCR monofonte générique . . . 101
8.2.1 Objectifs . . . 101
8.2.2 Génération de masques . . . 102
8.2.3 Algorithme de reconnaissance . . . 102
8.2.4 Optimisations . . . 103
8.2.5 Expériences . . . 104
8.2.6 Perspectives . . . 105
8.3 Reconnaissance de fontes . . . 106
8.3.1 Méthodologie . . . 106
8.3.2 Expériences . . . 106
8.4 Segmentation en mots . . . 107
8.4.1 Méthodologie . . . 107
8.4.2 Expériences . . . 107
8.5 Module d'apprentissage pour ScanWorX . . . 108
8.5.1 Méthodologie . . . 108
8.5.2 Expériences . . . 110
8.6 Post-correction d'OCR . . . 111
8.6.1 Méthodologie . . . 111
8.6.2 Expériences . . . 112
8.7 Convertisseur PostScript . . . 112
8.7.1 Méthodologie . . . 112
8.7.2 Expériences . . . 113

9 Modèles d'évaluation en reconnaissance assistée 115
9.1 Facteurs de coûts . . . 116
9.1.1 Coûts de reconnaissance et approches classiques . . . 116
9.1.2 Influence de l'utilisateur . . . 117
9.2 Formalisation . . . 117
9.2.1 Modèles de session de reconnaissance . . . 118
9.2.2 Stratégies d'organisation d'une session . . . 118

9.2.3 Modèle de coût pour les interventions de l'utilisateur . . . 119
9.3 Simulations . . . 121
9.3.1 Modélisation des erreurs de reconnaissance . . . 121
9.3.2 Observations . . . 122
9.4 Expériences . . . 125
9.4.1 Méthodologie . . . 125
9.4.2 Evaluation des coûts avec *ApOFIS* . . . 125
9.4.3 Evaluation des coûts avec ScanWorX . . . 126

10 Conclusions, perspectives et bilan **129**
10.1 Résumé des contributions scientifiques . . . 130
10.2 Leçons à tirer . . . 130
10.3 Critiques et extensions . . . 131
10.4 Bilan général . . . 132

Bibliographie **135**

A Couplage d'environnements de programmation **145**
A.1 Démarche proposée . . . 145
A.2 Agents – Un protocole général . . . 145
A.3 Expression dépendante du langage de programmation . . . 146
A.4 Spécification du noyau de coordination . . . 146
A.5 Exemple d'utilisation . . . 148
A.6 Implémentation . . . 149
A.7 Concept des jumeaux . . . 149
A.8 Perspectives . . . 150

B Compléments sur notre OCR monofonte générique **153**
B.1 Paramètres de configuration . . . 153
B.2 Utilitaires . . . 153
B.3 Observations . . . 154

C Simulation de lentilles magiques en Tcl-Tk **159**
C.1 Problématique et objectifs . . . 159
C.2 Solution proposée . . . 159
C.3 Exemple . . . 160
C.4 Implémentation . . . 161
C.5 Critique et extensions . . . 161

Abstract

This thesis addresses the question of document recognition with an assisted perspective, advocating an adequate combination between human and machine capabilities. Our contributions tackle various aspects of the underlying software architecture.

Both a study of existing systems and a projection on some future applications of document recognition illustrate the need of cooperative environments. Several mechanisms are proposed to drive the human-machine dialog, or to make the recognition systems able to improve with use.

The various data involved in a recognition system are organized in a modular and homogeneous way. The whole information is represented using the DAFS standard format. In our proposition, the control is decentralized according to a multi-agent modelling. This conceptual scheme is then simulated on our development platform, using concurrent, distributed, and multi-languages programming. An expressive solution is proposed for the coupling between the application kernel and a graphical user interface. A prototype is realized to validate the whole architecture.

Our software architecture takes advantage of the typographical know-how, through the use of a standardized font management support. This integrated approach lets us enhance the ergonomy, extend the possible use of the recognition results, and redefine some recognition techniques. A few innovative analyzers are described in the field of optical character recognition, font identification, or segmentation. The first experiments show that our simple methods behave surprisingly well, with respect to what can be expected from the state of the art.

Besides, we bring a contribution to the problem of measuring the performance of cooperative recognition systems, through the introduction of a new cost model. Our notations are able to describe assisted recognition scenarios, where the user takes part in the process, and where the accuracy is modified dynamically thank to incremental learning. Our cost model is used both in simulations and in experiments implying existing analyzers. The dynamic aspects of assisted systems can then be observed.

Keywords

- Document recognition
- Electronic document management – Documents formats and conversions
- Interactive environments – Human-machine cooperation
- Software architecture
- Multi-agents systems – Distributed artificial intelligence
- Concurrent and distributed programming
- Programming languages coupling
- Digital typography – Software support for font management
- Document image analysis methods – OCR – Segmentation – Font identification – Incremental learning
- Performance evaluation – Cost model

Résumé

Cette thèse aborde la reconnaissance de documents suivant une approche assistée, qui vise à exploiter au mieux les compétences respectives de l'homme et de la machine. Nos contributions portent notamment sur les questions d'architecture logicielle soulevées par la mise en oeuvre de systèmes de reconnaissance de documents.

Les avantages d'un environnement coopératif sont motivés par une analyse critique des systèmes actuels, et une projection sur les futures applications de la reconnaissance de documents. Diverses propositions concrètes sont émises sur la conduite du dialogue homme-machine, ainsi que sur les possibilités d'amélioration à l'usage.

L'inventaire des données à gérer dans un système de reconnaissance est organisé de façon modulaire et homogène, et représenté à l'aide du format standard DAFS. Sur le plan du contrôle, le système est décomposé selon une modélisation multi-agents. Cette découpe conceptuelle est alors simulée dans notre plateforme de développement, qui repose sur la programmation concurrente, distribuée, et multi-langages. Une solution expressive est proposée pour le couplage entre le noyau de l'application et l'interface graphique. Le prototype qui a servi à valider l'architecture est présenté.

Notre architecture logicielle encourage l'exploitation du savoir-faire typographique, par l'intermédiaire d'un support de fontes standardisé. Ce rapprochement entre les deux disciplines profite à la fois à l'ergonomie, à la valorisation des résultats de reconnaissance, et aux méthodes d'analyse automatiques. Nous présentons une poignée d'analyseurs originaux, pour des tâches de reconnaissance de caractères, d'identification des fontes, ou de segmentation. Les expériences conduites en guise de première évaluation démontrent l'utilité potentielle de nos outils d'analyse.

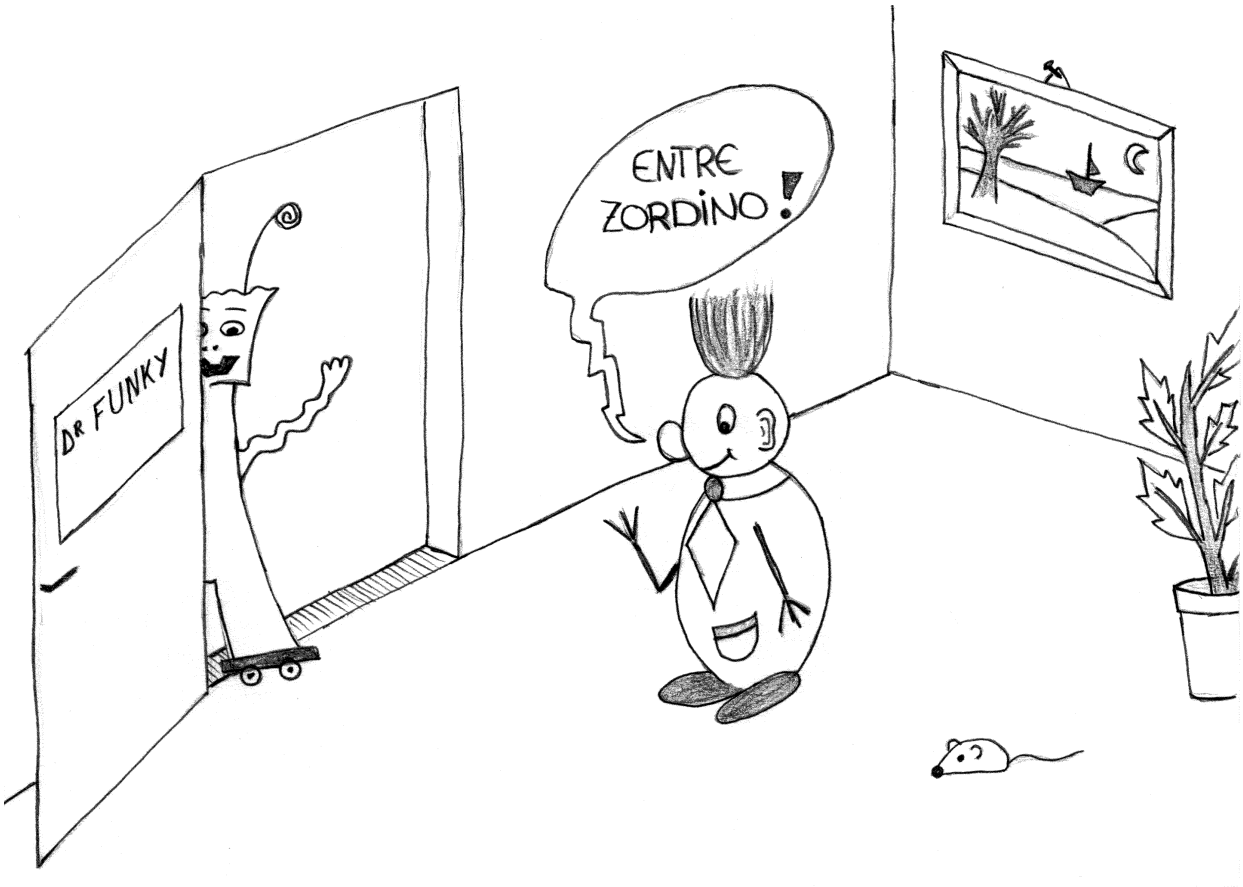
Par ailleurs, une contribution est apportée au problème de l'évaluation des performances de systèmes de reconnaissance assistée, avec l'introduction d'un nouveau modèle de coûts. Celui-ci intègre l'influence du comportement de l'utilisateur, de même que l'amélioration des performances liée au phénomène d'apprentissage incrémental. Notre modèle de coûts est utilisé dans des simulations, ainsi que dans des expériences mettant en jeu des analyseurs existants. Les observations mettent en évidence la dynamique particulière des systèmes assistés par rapport aux approches entièrement automatiques.

Mots-clés

- Reconnaissance de documents
- Gestion électronique de documents – Formats et conversions de documents
- Environnements interactifs – Coopération homme-machine
- Architecture logicielle
- Systèmes multi-agents – Intelligence artificielle distribuée
- Programmation concurrente et distribuée
- Couplage de langages de programmation
- Typographie numérique – Support logiciel de gestion de fontes
- Méthodes d'analyse d'images de documents – OCR – Segmentation – Reconnaissance de fontes – Apprentissage incrémental
- Evaluation des performances – Modèles de coûts

Chapitre 1

Introduction



Si l'informatique porte sur le traitement de l'information et ses nombreux visages, la *reconnaissance de documents* est une sous-discipline qui se préoccupe d'extraire l'information de l'un de ses supports les plus généraux, à savoir les documents écrits. La présente thèse est consacrée à deux aspects particuliers de la reconnaissance de documents. Premièrement, nous étudions les conséquences d'une approche *assistée*, qui exploite au mieux les compétences respectives de l'homme et de la machine. Deuxièmement, nous nous attaquons aux problèmes d'*architecture logicielle* soulevés par la réalisation d'un système de reconnaissance. L'étude se caractérise en outre par le souci de rassembler le savoir-faire impliqué dans des sous-problèmes très variés.

1.1 Problématique de la reconnaissance de documents

Depuis une vingtaine d'années, l'ordinateur a considérablement amélioré la gestion des documents écrits, que ce soit pour l'édition de texte ou de figures, la publication, le stockage ou la diffusion. En complément à la gestion documentaire classique, l'analyse d'images de document propose des moyens automatiques pour interpréter les informations véhiculées à partir du rendu visuel.

1.1.1 Généralités sur la gestion de documents

La notion de *document* recouvre toute forme d'information persistante. A ce titre, les documents sont omniprésents dans toute entreprise humaine en général, et dans le monde informatique en particulier. On parle de *documents numériques* lorsque les informations sont directement accessibles par l'ordinateur. La reconnaissance de documents porte uniquement sur des documents numériques, p. ex. l'image résultant de la saisie au scanner d'une feuille de papier.

Les documents peuvent transporter des informations de nature variée, et la catégorisation est loin d'être triviale. On distingue les documents écrits (représentant de l'information imprimée ou manuscrite), audio (représentant du son), ou vidéo (représentant des animations). Mais la technologie multi-média, actuellement en plein essor, tend à intégrer ces différentes formes. De plus, la notion d'hypertexte oblige à considérer des formes interactives de documents. Dans ce sens, la différence entre document et programme informatique devient de plus en plus floue. Nos travaux portent sur les *documents imprimables*, que nous définissons comme ceux qui sont adaptés à une lecture sur forme papier.

Au cours de l'existence d'un document imprimable, les informations contenues peuvent être codées sous différentes formes. Bien qu'il soit difficile de caractériser les sortes de structuration en toute généralité, la distinction suivante est couramment admise [90, 178] et largement applicable :

- La *forme logique* est destinée à la manipulation du message véhiculé par le document. Elle reflète le contenu sémantique, conformément aux intentions de l'auteur ou à la compréhension du lecteur. Par exemple, une base de données bibliographique au format BibTeX [120] est une forme logique, puisque les différents champs sont désignés par une étiquette symbolique (auteur, titre, etc.). Les formats typiques adaptés à la structuration logique comprennent SGML [80] et Thot [173]; d'autres formats comme LaTeX [120] offrent un support moins sophistiqué.
- La *forme physique* est destinée à la présentation visuelle des informations contenues dans le document. Elle reflète la mise en pages, conformément au point de vue du typographe. Parmi les formats qui codent la structure physique, on trouve PostScript [1] ou PDF [2].
- La *forme image* est destinée à la lecture oculaire du document. Elle ne fait que représenter directement la surface de chaque page, telle qu'elle apparaît à l'écran ou sur papier. TIFF est un format d'image très utilisé, qui convient pour les documents noir/blanc, à niveaux de gris, ou en couleur.

La figure 1.1 illustre les transformations de documents, à la fois dans le cycle normal de production et, dans le chemin inverse, en reconnaissance. L'édition porte sur la forme logique. La forme physique est obtenue par une étape de formattage. Le document est restitué en tant que collection d'images, prêtes à être affichées ou imprimées. Chacune de ces étapes est régie par une série de procédures prédéterminées, qui se prêtent à une automatisation.

La reconnaissance de documents est plus complexe que la production, dans la mesure où il faut remonter dans le degré d'abstraction à partir d'une donnée brute, sans structure. Différentes techniques sont nécessaires pour segmenter les pages de documents en blocs, par exemple pour délimiter les colonnes, les figures, ou les lignes d'un paragraphe. Sur les portions occupées par du texte, des méthodes permettent de reconnaître les chaînes de caractères, ou d'identifier les fontes. Tous ces résultats servent à reconstituer, au moins partiellement, la structure physique. D'autres traitements s'appliquent ensuite pour interpréter ce contenu en termes logiques, et pour extraire la signification des informations, telle qu'elle est perçue dans l'esprit du lecteur.

Plusieurs raisons expliquent pourquoi les documents ne sont pas systématiquement disponibles sous forme logique. D'une part, la structure dépend de l'utilisation visée : un même document admet plusieurs structures logiques utiles, selon l'usage qu'on attribue aux informations. D'autre part, la forme logique n'est pas directement adaptée à la lecture humaine, qui nécessite un support visuel comme l'écran ou le papier[1]. Pour lire de l'information disponible sous forme logique, il faut lui faire subir des transformations, en utilisant l'appareillage logiciel nécessaire au formattage et à la restitution. Par ailleurs, la distribution de la forme éditable des documents est freinée par des questions de droits d'auteur, à cause des risques de plagiat. En pratique, on constate une certaine confusion entre les formes logiques et physiques. Par exemple, les balises HTML [144] devraient aider à organiser la structure logique, mais le format est souvent utilisé pour encoder les attributs physiques.

[1] On nous prédisait l'avènement d'une société «zéro-papier».En 1995, un américain consommait en moyenne 341 kg de papier par an, soit une augmentation de 500% en quarante ans...

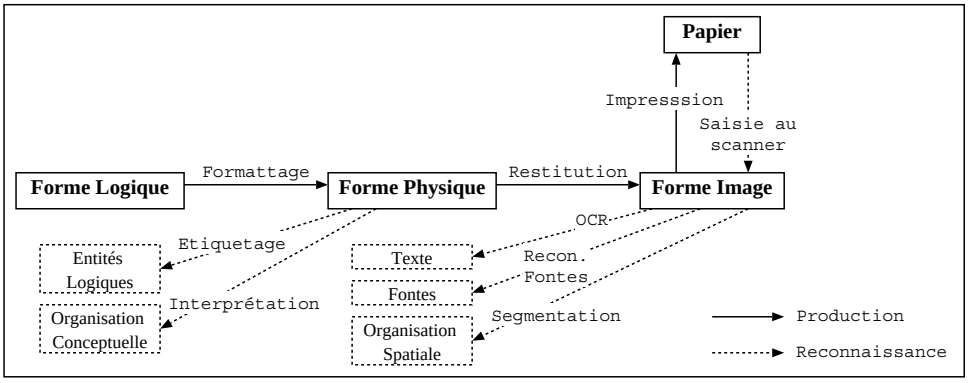


Figure 1.1 : Documents – Cycles de production et de reconnaissance.

1.1.2 Analyse d'images de documents

L'analyse d'images de documents est une discipline qui étudie les possibilités algorithmiques de reconstituer une information structurée à partir de la forme visuelle brute. On y propose des méthodes pour segmenter les images de documents, reconnaître le texte et la fonte, interpréter la sémantique du contenu, identifier des objets dans des schémas, récupérer la structure des tableaux, rechercher des logos, vérifier automatiquement les signatures, etc. La section 2.1 présente un bref survol de l'état de l'art dans ce domaine. Il n'existe pas de système de reconnaissance universel, et en pratique, les prototypes opérationnels sont toujours dédiés à une application.

Très tôt dans l'histoire de l'informatique, des chercheurs se sont penchés sur les difficultés à automatiser le processus de la lecture d'un document écrit. Pendant longtemps, les efforts se sont concentrés sur la reconnaissance optique de caractères (en anglais OCR, pour Optical Character Recognition), ainsi que sur quelques domaines très spécifiques où une certaine automatisation était très souhaitable d'un point de vue économique (lecture de chèques, tri postal, acquisition de formulaires).

Au fil des années, la discipline s'est continuellement enrichie, profitant des progrès réalisés dans des domaines proches, comme le traitement du signal, l'analyse d'images, la reconnaissance des formes, ou l'intelligence artificielle. Les chercheurs se sont attaqués à une foule de sous-problèmes, explorant autant les fondements théoriques que les approches empiriques. Chaque tentative d'application a mis en évidence de nouvelles difficultés pratiques, et des moyens pour les contourner.

Aujourd'hui, l'analyse d'images de documents possède tous les atouts d'un domaine scientifique intensif et bien organisé. Le savoir-faire accumulé est riche et varié. La technologie se concrétise par des prototypes de recherche opérationnels et des logiciels commerciaux. Chaque année, plusieurs colloques internationaux réunissent industriels et académiciens pour discuter les dernières innovations :

- International Conferences on Document Image Analysis and Recognition (ICDAR 91, 93, 95, 97);
- International Workshops on Document Analysis Systems (DAS 94, 96, 98);
- Annual Symposium on Document Analysis and Information Retrieval (SDAIR 91-96);
- Conference Internationale Francophone sur l'Ecrit et le Document (CIFED 97, précédemment CNED 90, 92, 94, 96).

Des ouvrages de synthèse [39, 14, 162] permettent de se familiariser avec l'état des connaissances actuelles. Depuis peu, on trouve une revue dédiée explicitement à la reconnaissance de documents : International Journal of Document Analysis and Recognition (IJDAR). Elle vient s'ajouter aux autres revues internationales acceptant régulièrement des papiers sur le sujet : IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI), Journal of the Pattern Recognition Society, Pattern Recognition Letters, IEEE Trans. on Image Processing, International Journal of Pattern Recognition and Artificial Intelligence. Des collections d'images de documents sont publiées afin de faciliter les comparaisons entre systèmes de reconnaissance : University of Washington (UW) [127], National Institute of Standards and Technology [73] (NIST). Des formats de données (DAFS [174, 62]) et des librairies ont été développés expressément pour supporter le processus de reconnaissance.

Face au foisonnement des travaux en reconnaissance de documents, notre groupe de recherche a décidé en 1994 de mettre son expérience au service d'un nouveau projet, qui constitue le cadre de cette thèse. Le projet

CIDRE [2] [16, 21, 17, 18, 19, 38, 37] (pour Cooperative & Interactive Document Reverse Engineering) reconsidère la problématique dans son ensemble, et cherche à revaloriser le rôle de l'utilisateur humain dans le processus de reconnaissance de documents. La section 2.3 présente en détail les objectifs du projet *CIDRE*.

1.2 Avenir des systèmes de reconnaissance

L'analyse d'images de document est encore une discipline en pleine expansion. Au moment de lancer un nouveau projet dans le domaine, il convient de cibler certaines niches prometteuses, et donc d'émettre des hypothèses quant à l'évolution des applications en reconnaissance de documents. Cette section présente une tendance qui pourrait et paraît se dégager pour l'avenir de l'ingénierie documentaire. Ce genre de discussion prospective n'est naturellement pas infaillible, mais apporte néanmoins un éclairage nécessaire sur les priorités que nous nous fixons. Dans la littérature [148, 149, 193, 210], d'autes projections critiques, notamment sur l'avenir de la reconnaissance de documents, ont déjà prouvé leur utilité pour guider la recherche actuelle.

Notre vision d'avenir s'articule autour de trois questions sur les futurs systèmes de reconnaissance : (i) quels besoins vont-ils satisfaire (cf. section 1.2.1), (ii) à qui seront-ils destinés (cf. section 1.2.2), et (iii) comment seront-ils utilisés (cf. section 1.2.3). Les trois aspects amènent conjointement à repenser le rôle de l'utilisateur humain.

1.2.1 Vers de nouveaux besoins

La reconnaissance de documents structurés a émergé à une période où beaucoup pensaient qu'à terme, la forme papier allait quasiment disparaître. Pour assurer la transition vers une gestion entièrement électronique, il fallait donc des utilitaires capables de récupérer les informations qui n'étaient disponibles que sur papier. Aujourd'hui, nous estimons que ce point de vue doit être corrigé sur deux points. D'une part, le papier est et continuera à être très utilisé. D'autre part, la prolifération des documents électroniques soulève de nouveaux problèmes de conversion de l'information, si bien qu'on est tenté de reformuler la problématique de reconnaissance de documents.

En matière de traitement de l'information, les documents utiles sont souvent accessibles sous une forme qui n'est pas adaptée à l'usage qu'on veut en faire. En général, plusieurs moyens peuvent servir à minimiser les problèmes de traduction entre formats. D'une part, la standardisation tend à réduire le nombre de formats utilisés, mais l'expérience a montré que la standardisation n'est jamais absolue. D'autre part, on peut concevoir des filtres dédiés à la conversion entre deux formats particuliers, mais la démarche doit être répétée pour tous les couples de formats. Nous proposons une troisième approche, basée sur deux idées : (i) l'image est considérée comme une *forme pivot*, qui peut facilement être générée depuis n'importe quel format; (ii) les techniques d'analyse d'images de documents aident à convertir vers la structure désirée l'information véhiculée par l'image. La figure 1.2 illustre cette nouvelle manière d'envisager l'utilité des systèmes de reconnaissance de documents. Le recours à l'analyse d'images a l'avantage d'être une voie très générale. La forme image est en effet une représentation universelle, car permettre une lecture visuelle du document est un objectif commun de tous les formats.

Nous parlons de *restructuration de documents* [4, 116, 68] lorsqu'on cherche à transformer la structure d'un document sans nécessairement changer de format de fichier. Par exemple, si on souhaite reprendre un article scientifique en tant que section d'un livre, il sera nécessaire de faire passer chaque section de l'article au rang de sous-section du livre. Sous le terme *réingénierie de documents*, nous englobons la reconnaissance classique de pages scannées, la reconnaissance d'images synthétisées, et la restructuration de documents.

Parallèlement à cette ouverture des systèmes de reconnaissance vers des formes plus générales d'entrées et de sorties, on devine de nouveaux besoins liés au cycle de vie des documents dans l'entreprise; en voici quelques exemples :

- *Bureautique.* Lors de la création et la mise à jour des documents dans un environnement bureautique, il est fréquent que les informations utiles ne sont pas disponibles sous la forme voulue. Nous pensons ici à la «petite» édition, celle qui englobe les rapports produits quotidiennement par un employé ou un service. Deux phénomènes, au demeurant souhaitables, contribuent pourtant à accroître en pratique les besoins en conversion de documents. Premièrement, l'édition des documents tend à être mieux structurée; le document est façonné selon l'utilisation prévue par son auteur, mais la structure choisie ne favorise pas forcément d'autres traitements. D'ailleurs, chaque éditeur (LaTeX [120], MsWord, Frame-

[2] ce projet est soutenu par le Fonds National de la Recherche Scientifique, subside no 21-42'355.94.

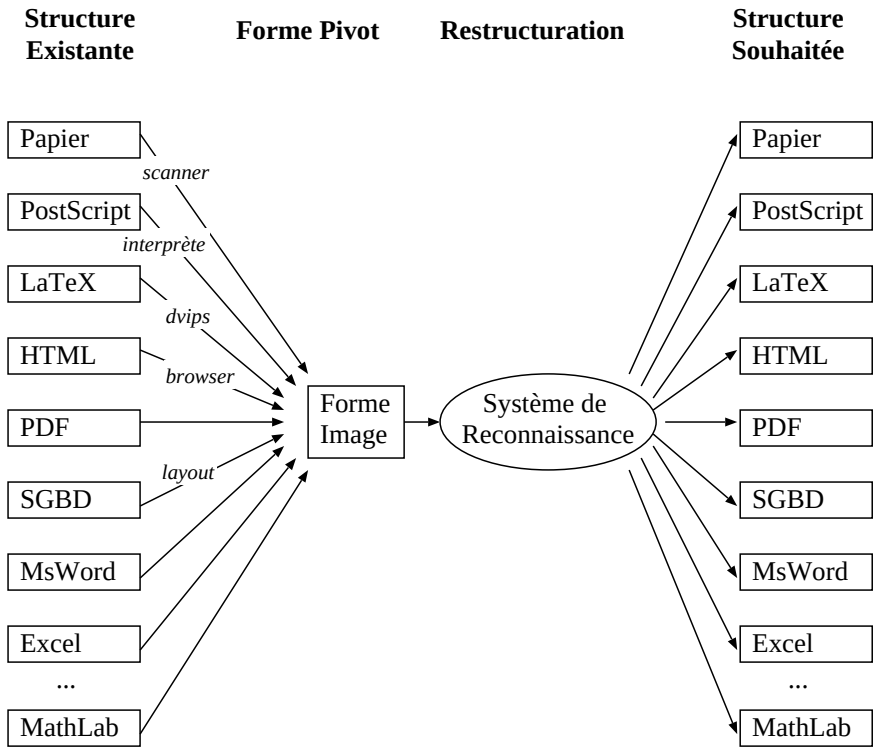


Figure 1.2 : Systèmes de reconnaissance au service des transformations de structures.

Maker etc.) propose ses propres moyens de structuration. Deuxièmement, les échanges d'information se multiplient, et avec eux les occasions de disposer d'une information dans une forme inappropriée, au moment de l'inclure dans un nouveau document.

- *WWW.* Le formidable essor du World Wide Web a engendré une nouvelle priorité dans la mise à disposition des informations. La conversion des documents vers HTML [144] n'est qu'un problème parmi d'autres. Souvent, l'organisation idéale d'un site n'est approchée qu'après plusieurs tentatives, ce qui implique des transformations successives [34]. La tendance à mettre un maximum de matière sur WWW incite à constituer de nouvelles bases de données, à partir d'informations disséminées dans l'entreprise. Enfin, la croissance exponentielle d'Internet soulève la question de l'indexation de la masse d'information disponible, dont la forme image constitue un dénominateur commun.

- *Archives.* La gestion des archives devient un sérieux problème suivant le type d'entreprise. Des dispositions légales imposent de conserver, parfois pour une longue durée, certaines catégories de documents, comme les pièces comptables ou la documentation technique (aviation, industrie nucléaire). La conservation sous forme électronique deviendra de plus en plus raisonnable[3], mais soulève le problème de l'indexation de cette masse de données. De plus, il arrivera qu'une partie des documents archivés soit réutilisée, nécessitant alors une structuration adéquate.

D'autres aspects de ces nouveaux besoins sont discutés dans la section 2.5, où nous présentons quatre familles d'applications qui devraient encore profiter des progrès en analyse d'images de documents.

1.2.2 Vers une démocratisation de la technologie

Malgré l'intensité des recherches autour de la reconnaissance de documents, il faut admettre que beaucoup de travaux restent à l'état d'expériences de laboratoire. Le transfert technologique est encore loin de fonctionner systématiquement dans notre discipline.

Lorsqu'on cumule les étapes, depuis l'analyse des besoins jusqu'à la maintenance, le coût de développement d'un système de reconnaissance destiné à l'exploitation se révèle prohibitif. Quand il faut extraire des informations à partir de documents papier, l'informatisation du processus ne s'avère rentable que pour quelques applications privilégiées (tri postal, lecture de chèques) qui portent sur un impressionnant volume de données.

[3] Un problème du stockage numérique tient à l'évolution rapide des supports électroniques; le papier a l'avantage de ne pas nécessiter d'appareillage spécial pour sa lecture.

Seule la reconnaissance de caractères est parvenue pour le moment à atteindre un usage plus vaste. D'une part, on a vu apparaître des logiciels commerciaux sur ordinateurs personnels pour reconnaître l'imprimé. D'autre part, l'avènement des agendas électroniques a banalisé la saisie du manuscrit en ligne.

Compte tenu des nombreux besoins encore insatisfaits, on devrait assister à une démocratisation de la technologie, et à l'apparition de petits utilitaires au service des problèmes bureautiques quotidiens. Nous prévoyons une évolution analogue à ce qu'on a pu constaté dans le domaine de la création de documents spécialisés, comme la composition musicale ou l'édition de pages HTML : les éditeurs de première génération étaient réservés aux professionnels. Puis sont apparues, en parallèle, des versions moins rigides à l'intention d'un plus large public.

A moyen terme, les organismes qui réceptionnent de gros volumes de documents (feuilles d'impôts, brevets, etc.) auront tendance à imposer le format électronique. C'est déjà le cas p. ex. pour la soumission d'articles à des revues scientifiques. Ces destinataires s'affranchissent par là des problèmes de structuration, puisque les documents arriveront sous une forme correcte. Nous prévoyons que les besoins de reconnaissance de documents apparaîtront par contre chez les expéditeurs, car (i) les standards diffèrent d'une entreprise à l'autre, et (ii) les individus qui composent les documents, eux, ne choisissent pas la forme de l'information qu'ils doivent exploiter. Cette nouvelle clientèle, plus individuelle, se caractérise par des besoins très variés : à chacun de ses partenaires externes correspond une nouvelle application de structuration de documents.

Du côté de l'offre, nous nous attendons à ce que certaines entreprises proposent sur Internet leur service de reconnaissance de documents : n'importe qui pourra leur soumettre des images de documents, et recevra plus tard, moyennant finance, une version correctement structurée. Que ce soit pour être utilisé par le client final ou par un fournisseur de services, le logiciel devra s'adapter à de nombreux types de documents. Il sera nécessaire de développer un noyau de programme capable de satisfaire plusieurs types d'applications. Dans ce but, les concepteurs devront s'attacher à résoudre deux sous-problèmes :

- Comment s'affranchir des trop nombreuses hypothèses sur les conditions d'utilisation du système? Le logiciel visé n'a pas à exclure a priori l'analyse d'un document, sous prétexte que les images sont à niveaux de gris, en couleurs, ou biaisées, que la résolution sort des normes, ou que les fontes sont exotiques. Les moyens techniques pour traiter les cas «spéciaux» existent déjà, mais cet effort de généralisation ne se justifie pas lorsqu'on construit un prototype de démonstration, ou un logiciel dédié à un client.
- Comment aider le client à configurer lui-même le logiciel pour ses besoins spécifiques? La notion de *modèle de documents* se réfère à une description de la structure commune à toute une classe de documents. Les systèmes paramétrés par un modèle de documents ont apporté un premier élément de réponse au problème de la configuration. Nous estimons que la solution mérite maintenant d'être étendue dans deux directions : (i) faciliter la création et la mise à jour d'un modèle, et (ii) englober dans la notion de document générique l'ensemble des paramètres de configuration.

Du côté de la demande, nous estimons qu'il faudra aussi faire des concessions sur les performances exigées. Il est rarement raisonnable de réclamer une solution informatique qui gère sans faute tout le processus de reconnaissance. Lorsqu'on leur promet une automatisation absolue, il ne faut pas s'étonner que les clients sont déçus, et renoncent à toute forme d'assistance logicielle. Le critère pour adopter un logiciel de reconnaissance doit se référer à la réduction des charges occasionnées par le procédé de reconnaissance. Peu importe qu'une partie de la main-d'oeuvre subsiste, du moment que la solution informatique améliore le rendement.

1.2.3 Vers une reconnaissance assistée

Les recherches en reconnaissance de documents ont répondu jusqu'à présent à un objectif d'automatisation. Ce but est souvent interprété de manière extrémiste, en tant qu'automatisation absolue. Notre expérience montre toutefois qu'une reconnaissance *entièrement* automatique est illusoire, sauf pour une poignée d'applications. Les logiciels de reconnaissance totalement automatiques ont un gros défaut : une fois installés, ils commettent toujours les mêmes erreurs. Pourtant, d'une manière ou d'une autre, chaque résultat d'un système de reconnaissance est finalement contrôlé, et ce contrôle en phase d'exploitation donne les plus précieuses indications sur la manière d'améliorer la précision des analyseurs.

La figure 1.3 rappelle que de la main-d'oeuvre est *de fait* impliquée en amont comme en aval du processus de reconnaissance proprement dit. En amont, il faut configurer le système, en lui indiquant notamment quelle structuration est attendue en sortie. En aval, un opérateur humain doit s'occuper manuellement des cas où la reconnaissance automatique a échoué. Nous prétendons que ces deux formes de main-d'oeuvre vont diminuer, un peu grâce à l'amélioration des performances, mais surtout grâce à l'apparition de systèmes interactifs d'aide à la reconnaissance de documents :

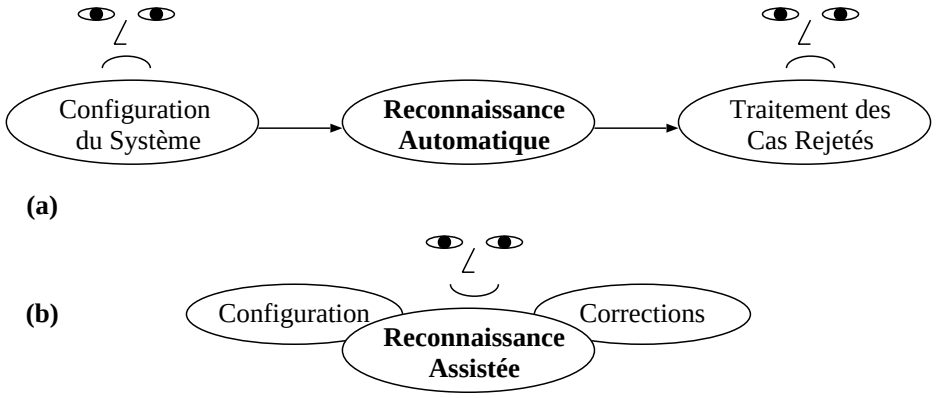


Figure 1.3 : Main-d'oeuvre en reconnaissance (a) automatique et (b) assistée.

- Effort de *configuration.* En général, recibler un logiciel est d'autant moins coûteux que cette opération se déroule hors de l'environnement de développement, donc proche de l'environnement d'utilisation. Par exemple, le moyen le plus convivial de spécifier une classe de documents serait d'en étiqueter manuellement quelques exemples significatifs. Or, aucun sous-ensemble d'échantillons n'est plus représentatif que ceux qui surgissent en cours d'exploitation. De plus, l'opérateur humain pourra ajuster la paramétrisation en fonction de la qualité des résultats observés. Bref, la capacité de réagir aux imprévus permet de pallier aux inévitables imperfections commises dans la configuration initiale.
- Effort de *correction.* Les erreurs de reconnaissance devraient être corrigées pendant la progression du traitement et le plus tôt possible, car on accorde ainsi au système une nouvelle chance de mener à bien l'analyse automatique. De plus, on peut compter sur les fonctionnalités d'apprentissage pour diminuer les occurrences de chaque type d'erreur rencontré. Par ailleurs, l'environnement interactif peut profiter du contexte d'analyse pour faciliter la correction manuelle, par exemple en proposant un choix de solutions alternatives.

Outre l'aspect comptable lié au rendement, il faut aussi reconnaître que, considérées séparément, ces deux formes de travail ne sont guère attractives. Le responsable de la configuration du logiciel ne perçoit que très indirectement l'influence de son expertise. De leur côté, les opérateurs chargés d'exploiter des résultats entachés d'erreur ont l'impression de subir les défauts de la machine, et sont consternés par les limites du système, notamment lors d'erreurs répétitives. Dans ces conditions, les discussions entre l'équipe de maintenance et les utilisateurs finaux risquent d'être houleuses. Dans un système interactif et adaptatif, l'opérateur humain a au moins la sensation de participer aux progrès de la situation, puisqu'il est capable de modifier le comportement du système qu'il utilise.

Nous espérons donc que les systèmes de reconnaissance apprendront à combiner de manière plus efficace les compétences respectives de l'homme et de la machine. Avec les futurs environnements assistés, la main-d'oeuvre impliquée sera intensive à chaque nouveau contexte d'utilisation, pour diminuer ensuite régulièrement à mesure que le nombre de documents traités augmente. Lorsque le logiciel est bien entraîné, les interventions humaines deviendront négligeables. Cette vision est un peu idyllique, et nous essayons dans cette thèse d'apporter des arguments plus tangibles sur la pertinence de l'approche assistée.

1.3 Objectifs de ce travail

Le présent travail se positionne dans le domaine de l'analyse d'images de documents. L'origine de nos réflexions se résume en une phrase : la place accordée aux utilisateurs humains dans les systèmes de reconnaissance de documents mérite d'être repensée. A partir de ce postulat, l'étude progresse davantage en largeur qu'en profondeur, car les implications sont très variées. Deux pôles d'intérêt se dégagent, comme l'annonce le titre de la thèse.

1.3.1 Conséquences d'une approche assistée

Le projet *CIDRE* défend l'idée d'un système de reconnaissance interactif, tandis que l'automatisation quasi-totale servait jusqu'alors d'hypothèse de travail en reconnaissance de documents. Notre étude porte sur les conséquences d'une approche centrée sur l'utilisateur, et couvre un large éventail de questions :

- Quelle représentation des modèles de documents serait adaptée à une acquisition interactive à travers des exemples? La section 2.3.4 présente les fondements de nos travaux sur la modélisation des documents. Ce sujet est développé dans un volet plus pointu du projet *CIDRE*, puisqu'il fait l'objet de la thèse de Rolf Brugger [36].
- Quelles applications profiteraient de l'approche interactive? La section 2.5 décrit quatre familles d'applications potentielles.
- Quel genre de dialogue homme-machine serait adapté à la reconnaissance de documents? Les sections 3.1 à 3.5 présentent plusieurs approches à la conduite de l'interaction.
- Comment organiser les données dans un système de reconnaissance assisté? Le chapitre 4 fait l'inventaire des informations à gérer, et propose une structuration adéquate.
- Comment modéliser l'architecture d'un système de reconnaissance interactif? Le chapitre 5 décompose le programme de reconnaissance au niveau conceptuel, en s'inspirant des systèmes multi-agents.
- Sur quelle plateforme logicielle faut-il implémenter une application de reconnaissance interactive? Les sections 6.2 à 6.4 présentent les caractéristiques de la plateforme que nous avons conçue.
- Comment relier l'interface homme-machine au noyau d'analyse? Les sections 5.3 et 6.3.3 proposent une solution qui favorise l'expressivité du codage.
- Comment concevoir des outils d'analyse qui s'intègrent bien dans un environnement assisté? Le chapitre 7 apporte quelques éléments de méthodologie, et montre par la même occasion les synergies possibles entre reconnaissance de documents et typographie.
- Comment réaliser des outils d'analyse simples et conduire les tests d'évaluation? Le chapitre 8 discute les analyseurs que nous avons développés.
- Comment évaluer les performances d'un système de reconnaissance? Les sections 9.1 et 9.2 montrent comment on pourrait intégrer l'utilisateur dans les modèles de coûts.
- Peut-on estimer la supériorité de l'approche assistée? Les sections 9.3 et 9.4 présentent les éléments de réponse que nous avons tirés de simulations, et d'expériences avec des outils d'analyse.

1.3.2 Architecture logicielle pour la reconnaissance assistée

Nous désignons sous le terme d'*architecture logicielle* l'organisation fonctionnelle et structurelle de l'ensemble d'un programme. Au niveau de la conception, il faut identifier les composants principaux du système, spécifier leurs rôles respectifs, et décrire les relations qu'ils entretiennent entre eux. Sur le plan de l'implémentation, la modélisation se concrétise par un ensemble de choix : environnements de programmation, organisation des sources, structures de données, etc. L'architecture logicielle aborde aussi la réutilisation de composants disponibles ou la compatibilité avec des standards existants. On voit que l'architecture logicielle touche la globalité d'un système, et se manifeste par de multiples facettes : représentation des données, interface graphique, outils d'analyse, notion de solution courante, découpe modulaire, moteur d'exécution, etc. D'une manière générale, le génie logiciel s'intéresse de plus en plus aux problèmes d'architecture logicielle [140].

La mise en oeuvre des objectifs de *CIDRE* nécessite une révision de l'architecture logicielle sous-jacente. La quête de convivialité bouleverse la nature même du processus de reconnaissance. Dans une approche automatique, le déroulement des opérations est déterminé à l'avance, et la conception du système peut adopter une optique purement algorithmique. Dans une approche assistée au contraire, le fonctionnement du programme est conditionné par les interventions de l'utilisateur, et le schéma de contrôle doit tenir compte de cette dimension interactive.

Il s'agit donc de concevoir une architecture qui intègre l'utilisateur en tant que participant au processus de reconnaissance, et non pas seulement dans l'introduction des données en entrée ou la visualisation des résultats en sortie. De plus, l'intensité des calculs impliqués en reconnaissance d'images impose certaines contraintes architecturales. Finalement, afin de favoriser la réutilisabilité et l'extensibilité, l'architecture doit tenir compte des pratiques usuelles en reconnaissance de documents, notamment en ce qui concerne les langages de programmation.

L'architecture logicielle est devenu notre thème central pour deux raisons au moins. Premièrement, c'est un élément déterminant dans la réussite d'une application de reconnaissance interactive, et souvent négligé au profit d'une optimisation des analyseurs automatiques. Deuxièmement, le projet *CIDRE* vise dans un premier temps le développement d'une plateforme logicielle générale pour la réingénierie de documents. Cette plateforme doit être conçue pour accueillir ensuite différentes applications.

On peut distinguer plusieurs types d'architectures logicielles, suivant les concepts dominants : analyse orientée objet, modèle du tableau noir, systèmes experts, etc. Pour notre part, nous avons choisi de nous inspirer des systèmes multi-agents. Notre approche repose sur des principes fondateurs de l'intelligence artificielle distribuée, tels que l'autonomie, la localité, la communication, ou le non déterminisme.

1.3.3 Intégration de savoir-faire

En marge du cahier des charges, notre étude apporte une contribution au problème de l'*intégration technologique*. Il faut bien constater en effet que les projets de recherche sont souvent très spécialisés, ce qui porte préjudice à la diffusion du savoir-faire entre disciplines. A force de se cantonner dans des sujets de plus en plus pointus, les scientifiques risquent de passer à côté des synergies indispensables à certaines percées. Nous ressentons fortement ce phénomène au sein même de la reconnaissance de documents. Par exemple, les préoccupations diffèrent passablement entre un chercheur spécialisé dans les algorithmes de reconnaissance du manuscrit en ligne, un expert en structures de documents, et le développeur d'un système industriel pour l'acquisition automatique de formulaires.

Nous utilisons le terme général de *savoir-faire* pour englober toutes les connaissances, théoriques et surtout pragmatiques, impliquées dans la réalisation d'un objectif. Construire un système de reconnaissance de documents est justement une activité qui requiert des aptitudes étendues : organiser l'acquisition des images, assimiler les algorithmes d'analyse, maîtriser un environnement de programmation, saisir le contexte de l'application, connaître des éléments de typographie, piloter la gestion de fontes du système informatique, appliquer les critères ergonomiques, concevoir des expériences sur une base statistique, mesurer la performance, déverminer, optimiser, etc. En grande partie, ce savoir-faire est acquis par expérience, et n'est pas expliqué directement dans la littérature.

En explorant l'ensemble des conséquences de l'approche assistée et en mettant l'accent sur l'architecture logicielle, ce travail renonce à figurer à la pointe d'un sujet bien délimité, mais cherche à garder la vue d'ensemble de la problématique. Nous essayons de prendre du recul vis-à-vis de différentes techniques informatiques, et nous considérons l'intégration de savoir-faire comme un sujet à part entière.

L'analyse d'images de documents a toujours formé un domaine interdisciplinaire par nature, puisqu'elle emprunte à la fois au traitement d'image, à la reconnaissance des formes, l'intelligence artificielle, l'édition de documents, ou le traitement de la langue naturelle. Depuis quelques années, la technologie s'éparpille de plus en plus dans des sous-disciplines de la reconnaissance de documents. Notre travail donne l'occasion de rassembler une partie du savoir-faire : techniques d'apprentissage, algorithmes d'analyse, structures de documents, standardisation, modèles de coûts. Nous avons également été amenés à intégrer des connaissances issues de domaines variés : programmation distribuée, programmation concurrente, interopérabilité, interfaces homme-machine, ergonomie, génie logiciel, génie documentaire, typographie.

L'effort d'intégration technologique aboutit parfois à rapprocher des disciplines. C'est ainsi que nous avons pu unifier la notion de fonte, sur laquelle différaient les points de vue de la typographie et de la reconnaissance de texte (cf. section 7.3). Cela a permis d'améliorer l'interface graphique, de mieux valoriser les résultats de reconnaissance, et même de proposer de nouvelles méthodes d'analyse.

1.4 Organisation en chapitres

La thèse est organisée en dix chapitres et trois annexes.

Le chapitre 2 situe le *contexte du travail*. Il fait le point sur l'état de l'art en reconnaissance de documents, et expose les ambitions du projet *CIDRE*. Nous introduisons le concept de reconnaissance assistée, ainsi que quelques champs d'application inédits.

Le chapitre 3 discute l'*interface homme-machine*. A la lumière de l'état de l'art en interaction homme-machine, nous reconsidérons le rôle de l'utilisateur dans une session de reconnaissance, en décrivant des formes possibles de dialogues.

Le chapitre 4 est consacré à la *gestion des données*. Nous établissons l'inventaire des informations que l'architecture devra maintenir, et une solution concrète, basée sur le format DAFS, est présentée.

Le chapitre 5 décrit une *modélisation multi-agents*. Nous présentons les motivations de l'intelligence artificielle distribuée, puis nous essayons d'organiser l'architecture d'un système de reconnaissance selon une approche multi-agents.

Le chapitre 6 présente une *plateforme d'implémentation*. La solution proposée permet de concrétiser la découpe multi-agents, et de profiter des progrès en programmation concurrente, distribuée, interactive, et multi-langages.

Le chapitre 7 se concentre sur les *techniques d'analyse*. Nous mettons en évidence une tendance dans la conception d'analyseurs respectant le contexte informatique en général, et l'approche assistée en particulier.

Le chapitre 8 présente les *outils que nous avons réalisés*. Nous exposons nos propres développements d'analyseurs, et rapportons des observations sur leur expérimentation.

Le chapitre 9 aborde la *modélisation des coûts*. Nous esquissons une nouvelle approche adaptée à l'évaluation de systèmes assistés et adaptatifs. Cette proposition est étayée par une notation ad hoc, des simulations, et des expériences pratiques.

Enfin, le chapitre 10 expose les *conclusions du travail*. Nous passons en revue les contributions scientifiques apportées par notre étude, ainsi que les perspectives pour des travaux ultérieurs.

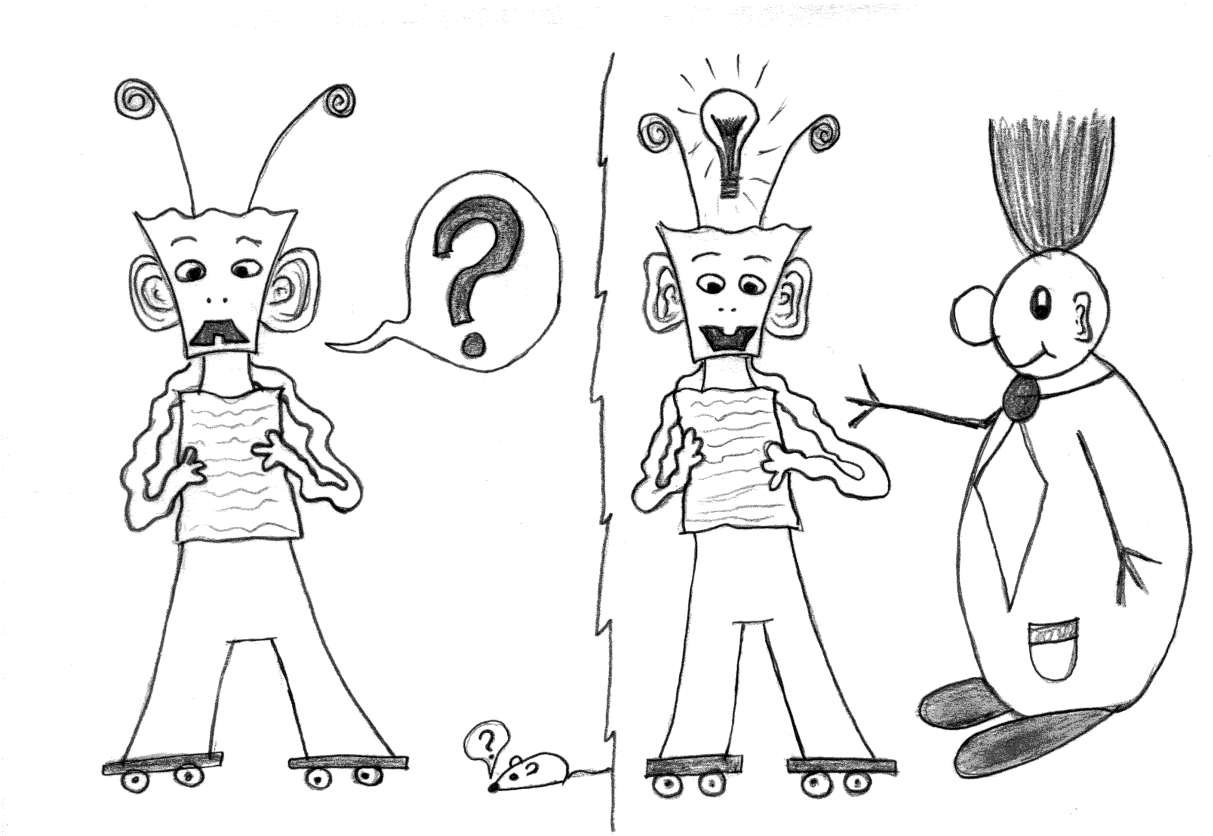
L'annexe A décrit une solution générique au *couplage de langages de programmation* hétérogènes. Cette proposition est issue d'une généralisation de notre architecture qui intègre au noyau d'analyse une interface graphique écrite en Tcl-Tk.

L'annexe B apporte quelques compléments d'information sur le développement de notre *OCR monofonte*.

L'annexe C discute un moyen de simuler le concept de *lentille magique* dans le langage de programmation Tcl-Tk.

Chapitre 2

De la reconnaissance automatique à la reconnaissance assistée



Ce chapitre introduit les motivations du projet *CIDRE*. Nous commençons par survoler l'état de l'art en reconnaissance de documents. La section 2.2 expose les critiques que nous formulons envers les systèmes de reconnaissance actuels. Ces réflexions ont donné naissance au projet *CIDRE*, dont les objectifs sont énoncés dans la section 2.3. Comme le reste de la thèse est centré sur des problèmes généraux comme l'architecture logicielle, il nous a paru important de montrer le chemin vers des prototypes appliqués. Dans la section 2.4, nous indiquons quelques sujets de réflexion qui peuvent servir à amorcer le processus de conception, quelle que soit l'application visée. Par ailleurs, quatre applications potentielles sont présentées dans la section 2.5.

2.1 Survol de l'état de l'art

L'analyse d'images de documents est maintenant une discipline bien établie. Cette section est destinée à donner au lecteur une idée des innombrables travaux accomplis dans ce domaine. La première partie rappelle les principales étapes dans le processus de reconnaissance. La section 2.1.2 fait le point sur les problèmes de reconnaissance qui sont considérés comme résolus. La section 2.1.3 montre l'étendue des travaux en cours,

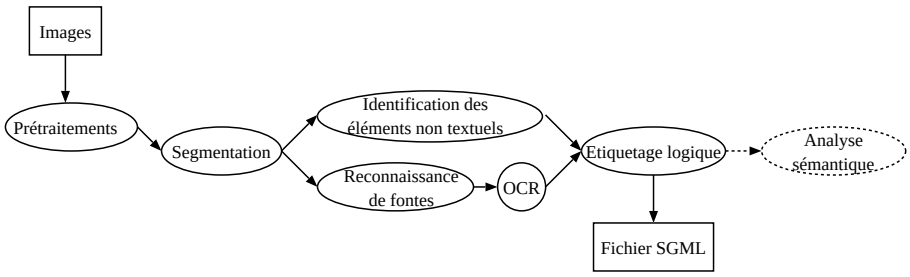


Figure 2.1 : Chaîne de traitements conventionnelle, sous forme de pipeline rigide.

qui portent sur une variété de catégories d'éléments susceptibles d'apparaître dans les documents. La section 2.1.4 énumère quelques systèmes complets qui sont opérationnels et intègrent toutes les étapes d'analyse. Enfin, la section 2.1.5 présente des projets en cours qui ont des affinités particulières avec notre propre projet *CIDRE*.

2.1.1 Etapes du processus de reconnaissance

La figure 2.1 illustre les principales étapes que l'on retrouve dans la plupart des systèmes de reconnaissance.

Les images de documents sont d'abord prétraitées, par exemple pour en améliorer la qualité. Ensuite on procède à une segmentation pour identifier les unités qui composent chaque page. Dans les parties textuelles, on cherche à détecter les principales fontes, puis on procède à la reconnaissance de caractères (OCR). Les éléments non textuels sont analysés par des outils spécialisés.

Mais la reconnaissance de documents ne se limite pas aux tâches d'OCR et de segmentation. En général, il est important de découvrir la structure logique sous-jacente, et d'identifier par exemple le nom des auteurs, les en-têtes, les notes de bas de page, les titres ou les mots-clés. C'est pourquoi le système de reconnaissance cherche à identifier des unités logiques et à leur associer des étiquettes. Enfin, certains travaux essaient de deviner la sémantique de l'information véhiculée par le document, par exemple en recourant au traitement de la langue naturelle. Mais on peut facilement imaginer les imperfections d'une telle entreprise, qui se situe à la limite de la reconnaissance de documents.

2.1.2 Problèmes pratiquement résolus

Cette section aborde deux questions. Quand faut-il considérer qu'un logiciel de reconnaissance est totalement fiable? Quels sont les problèmes de reconnaissance qui semblent résolus à l'heure actuelle?

Fiabilité et rentabilité Avant d'évoquer les cas où la reconnaissance de documents donne de bons résultats, il faut s'en tenir à un critère de qualité. Nous considérons ici qu'un analyseur est *fiable* sur un jeu de données lorsqu'il engendre un nombre d'erreurs équivalent ou inférieur à ce qu'on obtient habituellement par traitement manuel. Il n'est pas raisonnable d'exiger l'absence totale d'erreur, car l'humain introduit lui-même des erreurs, comme les fautes de frappes. Par ailleurs, on peut noter que l'interprétation d'une certaine donnée varie parfois selon les personnes, par exemple en *vérification de signatures*. Certains problèmes de reconnaissance sont donc effectivement ambigus. [58].

Même sans être fiable, le recours à un outil de reconnaissance peut s'avérer *rentable*, dans le sens où il engendre moins de coûts qu'une solution entièrement manuelle. En général, l'application exige des résultats finaux fiables, et on choisit un procédé semi-automatique. La plupart des méthodes d'analyse automatique sont capables d'associer une valeur de confiance aux résultats produits. Le logiciel peut ensuite décider de rejeter les résultats dont la confiance est inférieure à un certain seuil. La démarche habituelle consiste à choisir un *taux de rejet* [46] tel que les résultats soient fiables sur les documents acceptés. Les documents contenant des parties rejetées sont ensuite traités manuellement. C'est ainsi par exemple que la très forte demande pour automatiser le *tri postal* [109], la *lecture de chèques* [3], l'*acquisition de formulaires* [160] a pu être en partie satisfaite.

Situations considérées comme fiables Si l'on adopte notre critère de fiabilité pour juger les performances des analyseurs actuels, le bilan est plutôt terne. Naturellement, on peut atteindre une reconnaissance fiable lorsqu'on impose des contraintes très strictes sur la *production* des documents, comme par exemple pour des formulaires composés avec une fonte spécialement adaptée à l'OCR (OCR-A, OCR-B). Quand on ne

peut pas influencer la publication des documents, la reconnaissance de l'écrit est loin d'avoir atteint un niveau de précision quasi-absolu. Cela fait plus de 40 ans que des travaux sont consacrés à la *lecture optique de caractères* (OCR) [151]. De nombreuses solutions commerciales sont maintenant disponibles [29, 67]. Malgré tous ces efforts, ce n'est que lorsque les conditions suivantes sont réunies que l'OCR peut actuellement être considéré comme fiable, et afficher un taux de reconnaissance dépassant 99.9% :

- les images sont de très bonne qualité, de résolution suffisante (300 dpi, ou davantage avec des fontes au-dessous de 10pt), et contiennent uniquement du texte imprimé continu (p. ex. pages de roman);
- le texte utilise l'alphabet latin, formatté dans une fonte simple (pas de souligné, pas d'italique);
- le texte est conforme à une langue connue, et contient peu de noms propres ou de caractères spéciaux (lettrines, indices, exposants).

Bien peu d'applications peuvent donc être résolues de manière entièrement automatique, car les documents sont en réalité bien plus complexes.

2.1.3 Ce qu'on essaie de reconnaître

Cette section montre l'étendue du savoir-faire accumulé en analyse d'images de documents. Une grande variété de problèmes de reconnaissance a été étudiée. Certaines solutions ont atteint un degré de maturité élevé, sans toutefois offrir des résultats totalement fiables. Néanmoins, les techniques inventées permettent de rentabiliser une automatisation partielle de certaines applications.

De très nombreuses méthodes de segmentation ont été proposées pour reconnaître toute la *découpe physique* [10, 94] des pages de documents. Les analyseurs tentent de déterminer la nature des portions d'image détectées, par exemple les photographies, graphiques, tableaux, notes de bas de page, filets, etc. A part les problèmes de dégradation de l'image, les cas les plus difficiles concernent les mises en page complexes, telles que les magazines ou les journaux.

La reconnaissance de *fontes* a étrangement été négligée pendant longtemps, mais quelques travaux récents [215, 189, 141] ont suffi pour parvenir à des résultats qui semblent satisfaisants, quoi qu'on manque encore de recul.

Dans le cas du *texte imprimé*, les analyseurs reculent constamment les limites de la reconnaissance des caractères, en traitant par exemple des alphabets non latins [5, 60], et des images couleurs ou à faible résolution [211].

On ne compte plus le nombre annuel croissant de travaux consacrés à la reconnaissance de l'*écriture manuscrite* [117]. Les chercheurs s'attaquent à des problèmes de plus en plus complexes, comme l'écriture cursive, l'arabe, ou les idéogrammes orientaux. La technologie de la reconnaissance en ligne (c'est-à-dire exploitant l'aspect temporel du tracé) est maintenant embarquée dans les agendas électroniques.

Plusieurs approches ont été mises en oeuvre pour procéder à l'*étiquetage logique* [86] des documents. La plupart des prototypes sont paramétrés par une description formelle de la classe de documents, mais ce genre de description est très difficile à obtenir.

La structure particulière des *tableaux* [187] nécessite des méthodes de reconnaissance idoines, afin de récupérer le contenu des cellules et leur organisation matricielle. Mais les tableaux peuvent être arbitrairement complexes, ce qui laisse peu d'espoir d'inventer une solution définitive et universelle.

Les *formules mathématiques* [33] suivent des règles assez strictes, ce qui a permis d'inventer des algorithmes qui tentent de découvrir leur structure symbolique. Les difficultés sont nombreuses : relations spatiales variables, indices et exposants, alphabet très étendu, etc.

La reconnaissance de *dessins techniques* [201] (schémas mécaniques, électriques, dessins architecturaux) soulève encore des problèmes complexes. La demande pour des solutions informatiques est d'autant plus ressentie que les logiciels de conception assistée par ordinateur (CAO) se sont généralisés.

Parmi toutes les sortes de documents qui posent des défis particuliers pour la reconnaissance, on peut mentionner la *cartographie* [153, 209] (cartes géographiques, plans cadastraux) ou les *partitions musicales* [12].

En fait, chaque application de reconnaissance engendre un savoir-faire spécifique. En guise d'exemple, notre groupe de recherche a développé des compétences en reconnaissance des fontes [215, 213], en reconnaissance de structures physiques [10] et logiques [86, 38], en classification d'images de formulaires, ou en détection de pages ou de formulaires vides [177]. C'est pourquoi nous cherchons à mettre toute cette expérience au service du projet *CIDRE*.

2.1.4 Quelques prototypes complets

Cette section passe en revue quelques projets qui ont abouti à des prototypes opérationnels. Chacun à leur manière, les concepteurs ont donc franchi les obstacles qui se dressent entre des bribes de solutions isolées et le développement d'une application complète. Compte tenu de la richesse des travaux dans le domaine, la liste suivante n'est certainement pas exhaustive.

Gobbledoc [152] Au Rensselaer Polytechnic Institute, George Nagy a dirigé la mise au point d'un outil de reconnaissance appliqué aux articles scientifiques. Le système Gobbledoc exploite une structure de données basée sur les arbres X-Y, et adopte une approche essentiellement syntaxique pour l'étiquetage logique. Le modèle de document est représenté sous forme grammaticale. Le moteur d'analyse est plutôt conventionnel, mais l'architecture tient compte de la répartition des traitements sur plusieurs processeurs dans un réseau de machines hétérogènes. Le projet n'aborde pas le rôle de l'utilisateur, ni les questions d'apprentissage. Même si Gobbledoc est resté à l'état de prototype, il a eu une influence sensible sur la communauté scientifique.

InfoPortLab [26, 25] Chez Daimler-Benz, l'équipe de Thomas Bayer travaille sur un prototype ambitieux d'extraction d'information à partir de documents papier, essentiellement des documents commerciaux (lettres, factures). L'application étudiée porte sur l'acheminement automatique de courrier. Le processus de reconnaissance est divisé en modules : analyse d'image de document, analyse du texte structuré, catégorisation de texte, et analyse du texte continu. Le système vise à automatiser les tâches de routines, tandis qu'à travers une interface graphique, l'utilisateur peut s'occuper manuellement des documents rejetés (cas non prévus ou trop dégradés). La panoplie d'analyseurs, qui couvre tout le spectre des traitements, est très riche et représentative de l'état de l'art. Les indications sur le modèle de documents sont représentées dans des réseaux sémantiques avec le langage FRESCO. La compatibilité avec les formats standards comme SGML ne semble pas abordée. Une architecture à base de tableau noir assure la souplesse nécessaire, de sorte que le système n'est pas limité aux analyseurs existants, ni à une séquence d'étapes prédéfinie. Le système est reconfigurable, mais implique alors un apprentissage lourd portant sur un volume de données important.

OfficeMAID [56, 24] Au DFKI, l'équipe d'Andreas Dengel a mis au point le prototype OfficeMAID. L'application visée concerne l'automatisation du traitement des documents (lettres, formulaires) dans un service d'achat. Le système tient compte des contraintes organisationnelles et des flux d'activités adoptés dans l'entreprise. Cela permet notamment de dépasser la reconnaissance individuelle de chaque document, et de tisser des liens entre documents, par exemple pour former un dossier de tous les documents concernant une même commande. L'information extraite est dirigée par les buts. Bien que chaque étape ait été optimisée, l'architecture générale ne semble pas autoriser de retour en arrière dans les niveaux d'analyse. Les besoins de l'utilisateur sont respectés en tant que consommateur des résultats; toutefois, il ne participe pas à la session de reconnaissance. On peut entraîner l'étiquetage logique avec quelques documents commerciaux typiques, mais rien ne laisse supposer que le noyau de l'application peut facilement être adapté à un autre usage.

SPAM [192] Larry Spitz supervise le développement de SPAM, une application destinée à faciliter l'accès à une base de données d'articles scientifiques. Ce projet ne revendique pas de percée technologique, mais entend plutôt assembler le savoir-faire établi au service d'un programme simple et flexible, pour un usage quotidien et personnalisé. La base de donnée est alimentée avec des articles scannés, mais aussi sous forme électronique (LaTeX, PDF, PostScript, MIF). Le système procède notamment à une indexation par mots-clés, selon un lexique spécifique au domaine; ce lexique est tout d'abord estimé de manière statistique, puis il peut être mis à jour manuellement. SPAM génère pour chaque document une référence bibliographique au format BibTeX. L'utilisateur pilote l'environnement à travers une interface graphique écrite en Tcl-Tk.

Oscar-II [86] A Fribourg, Tao Hu a construit un prototype de reconnaissance de structure logique, illustré sur des ouvrages scientifiques et des textes de loi. L'effort a été porté sur l'algorithme d'étiquetage logique, paramétré par une description formelle de la classe de documents. Ce modèle [90], formalisé en grammaire attribuée, permet de représenter des structures jusqu'au plus haut degré de détail. L'incertitude est gérée au moyen de la logique floue, et le processus de reconnaissance est guidé par un algorithme de programmation dynamique. Par la suite, Tao Hu a utilisé une méthode similaire pour étiqueter les tables des matières dans des revues [87]. L'adaptabilité et l'interactivité sont totalement absentes de l'étude.

Graphein [44] Dans l'équipe d'Abdel Belaïd à l'INRIA de Nancy, Yannick Chenevoy a développé le système général Graphein, et l'a validé sur deux applications, l'une portant sur des notices bibliographiques, l'autre sur les articles d'une certaine revue. La modélisation des documents s'inspire d'ODA [7]. L'architecture

de Graphein, basée sur le tableau noir, repose sur le système multi-experts ATOME [122]. L'utilisateur n'est pas impliqué dans le processus de reconnaissance.

BAsCET [167] Dans la même équipe, François Parmentier a développé un système automatique de reconnaissance, appliqué aux références bibliographiques. L'innovation réside dans l'architecture, qui s'inspire des paradigmes d'intelligence artificielle : réseau de concepts, coopération entre agents, émergence. La stratégie d'analyse est non déterministe, car les différents traitements ne s'enchaînent pas selon une procédure prédéfinie, mais sont activés en fonction de l'évolution de la solution courante. Le modèle de documents maintient des informations statistiques; il est inféré automatiquement à partir d'une base de données de référence. L'approche choisie se révèle robuste, et permet de surmonter la variabilité des formats. L'étude est toutefois limitée à l'étape de reconnaissance logique, car le système prend en entrée le texte exact composant la référence bibliographique.

DocBrowse [95] Le système DocBrowse, maintenu chez MathSoft Inc., est un logiciel de gestion de bases de données d'images de documents. Ce prototype est orienté vers la recherche d'information et la navigation dans la base de données. L'utilisateur soumet une requête de recherche par mots-clés, ou encore visuellement en donnant un exemple de logo ou de signature. Le système ne cherche pas à reconnaître la structure logique, mais exploite au mieux les descripteurs visuels. L'architecture sépare les fonctionnalités, afin de tirer profit de paquetages qui ont fait leurs preuves, comme SPlus, Khoros ou ScanWorX. L'interface graphique est très conviviale dans la spécification ou le raffinement des requêtes, ainsi que dans la présentation des documents trouvés.

Manicure [198] A l'Université de Las Vegas, Kazem Taghva dirige le projet MANICURE. Il s'agit d'un système de traitement de documents destiné à la création d'une collection électronique de documents imprimés. Trois composants principaux sont développés : AutoTag attache des balises SGML à la structure géométrique fournie par l'OCR. PPSYS est un outil de post-traitement, qui vise à corriger un maximum d'erreurs d'OCR en tenant compte de la totalité des résultats de reconnaissance, ou de la matrice de confusions de l'OCR. Enfin, Rummage est un environnement interactif de correction semi-automatique. Le système tient compte des besoins en terme de recherche d'information, mais ne semble guère adaptatif.

PRASAD [123] Chez EDF, Philippe Lefèvre et ses collègues ont mis au point le système PRASAD, qui s'est révélé capable de reconstruire la structure physique de documents variés. Les données sont manipulées dans le format ODIL [124]. ODIL spécifie, sous forme d'une DTD (Document Type Definition) pour SGML [80], un langage permettant de décrire la mise en page des documents, même en présence de formules ou de tableaux. PRASAD offre une interface utilisateur pour permettre la vérification et la correction des résultats, mais n'a pas de capacités d'apprentissage. Le système fait collaborer trois OCR, et est capable de traiter les images à niveaux de gris. Rien n'est prévu pour faciliter l'adaptation à une nouvelle classe de documents, et la reconnaissance logique n'est pas abordée. La maintenance du produit a été cédée à l'entreprise PRITEC.

2.1.5 Quelques projets proches de *CIDRE*

Cette section évoque cinq projets récents qui, sous certains aspects, ont rejoint les objectifs de notre projet *CIDRE*. Chacun à sa manière, ces travaux montrent que l'architecture logicielle devient un sujet de première importance en reconnaissance de documents.

TABS [52] A l'Université d'Essex, Chris Cracknell travaille sur la plateforme TABS, dédiée au traitement, à l'analyse et à la compréhension d'images. TABS permet de réaliser des composants (outils d'analyse), puis de les intégrer de façon cohérente au sein d'un système complet. Le système adopte une approche multi-langages (C, C++, Tcl, voire d'autres langages), et exploite en particulier les avantages de Tcl-Tk. L'architecture, basée sur un tableau noir, permet de manipuler toutes sortes de représentations symboliques intermédiaires. Un support est notamment offert pour gérer des résultats sous forme de listes d'hypothèses ou de structures hiérarchiques. Par ailleurs, le noyau d'analyse est clairement séparé de l'interface. Les applications peuvent facilement être pilotées à travers une interface graphique. La plateforme définit la notion d'espace de travail, ce qui permet de sauvegarder sur disque l'état courant du système. Comme autre particularité, des utilitaires permettent de tracer les performances du système. Un prototype a été développé pour une application d'acquisition de formulaires.

DART [79] A l'Université de Nottingham, Andreas Hennig propose la plateforme DART pour la création d'une boîte à outils adaptée à la reconnaissance de documents. Le but est de faciliter le passage entre une

collection d'algorithmes spécialisés et un moteur d'analyse complet. DART est une architecture logicielle destinée à gérer l'intégration d'outils d'analyse. La structure générale d'un analyseur a été modélisée. L'architecture assemble sous forme de graphe un ensemble d'analyseurs, qui exécutent des traitements de façon asynchrone, et peuvent même être distribués sur un réseau de machines. La plateforme permet de modifier aisément la configuration des analyseurs et le schéma d'interconnexion. Les flux de données prévoient un mécanisme de rétroaction. Globalement, l'architecture se caractérise par une grande flexibilité, en phase de développement comme durant l'exécution. Le prototype a entre autres été utilisé dans une application de reconnaissance de documents manuscrits.

UW-ISL [128] A l'Université de Washington, Robert Haralick conduit un projet destiné à faciliter les échanges et les comparaisons au sein de la communauté de reconnaissance de documents. Après l'élaboration du format DAFS [174] et la production d'une base de données d'images de documents [127], les travaux se concentrent maintenant sur la constitution d'une boîte à outils dédiée à l'analyse d'images de documents. Cette plateforme offre à la fois une collection d'analyseurs compatibles avec DAFS, et un moteur d'exécution reconfigurable. Les premières applications visent la conversion d'images de documents vers des formats standards (RTF ou PDF). Un soin particulier est apporté à l'évaluation des algorithmes mis à disposition. En revanche, les problèmes d'interactivité ou d'apprentissage incrémental ne sont pas évoqués pour le moment.

CIME [126] Au Centre de Recherche Informatique de Montréal, Marc Lalonde travaille sur CIME, un environnement de compréhension d'images adapté au traitement de documents. Le système s'articule autour de bases de connaissances. Un langage est proposé pour modéliser les objets à reconnaître dans l'image, et sert à exprimer des propriétés sur la forme, la taille, ou les relations spatiales entre composants. CIME offre un gestionnaire de tâches qui permet au concepteur de définir des stratégies d'analyse pour affronter les problèmes de reconnaissance. L'environnement CIME définit aussi une interface graphique homogène, qui permet de configurer le système et de visualiser les résultats. Le prototype a p. ex. été appliqué pour la reconnaissance de symboles dans des cartes marines et des schémas électriques.

PS2HTML [169] A l'Ecole Polytechnique de Montréal, Benoît Poirier développe un environnement interactif de conversion de documents PostScript, vers le format HTML. Le principe consiste à interpréter le programme PostScript pour en extraire toutes les primitives géométriques, puis à réorganiser ce contenu selon les balises HTML, en collaboration avec l'utilisateur. De nombreuses heuristiques sont mises en oeuvre : p. ex., le regroupement de caractères en mots tient compte des métriques de la fonte courante, et de la table de crénage. Le système est robuste quant à la provenance des documents PostScript. L'architecture se veut particulièrement modulaire et extensible. Chaque module indépendant fouille la structure géométrique pour y détecter un certain type d'entité (en-tête, titre, liste). La reconstitution des tableaux est très bien soignée. La notion de classe de documents n'a pas encore été appliquée au prototype.

2.2 Limites des systèmes actuels

Dans cette section, nous mettons en évidence les points faibles qu'on trouve dans la plupart des travaux antérieurs en reconnaissance de documents. Cette réflexion sur les limites des systèmes actuels a servi de base pour dresser le cahier des charges du projet *CIDRE*.

2.2.1 Rôle de l'utilisateur

L'expérience des systèmes de reconnaissance existants nous apprend deux choses : (i) les erreurs de reconnaissance sont inévitables, compte tenu de l'imperfection des techniques d'analyse et de la variabilité des données d'entrée; (ii) rares sont les applications qui tolèrent la présence d'autant d'erreurs dans les résultats finaux. Cela signifie qu'en pratique, un opérateur humain est chargé de valider et corriger les résultats douteux, ainsi que de traiter manuellement les documents où la reconnaissance automatique a échoué. Un autre type de main-d'oeuvre est sollicité lors de la configuration du logiciel dans son contexte d'utilisation. Bref, on ne croit plus à une automatisation totale et parfaite de la reconnaissance de documents.

La grande majorité des travaux en reconnaissance de documents ne se soucient pas du rôle de l'utilisateur. L'interaction est généralement exclue d'avance, puisque l'objectif inconditionnel est une automatisation totale du processus de reconnaissance. Sous ce point de vue, il n'y a qu'un seul rôle souhaitable pour l'utilisateur humain, du moins en phase d'exploitation, c'est d'être complètement déchargé du travail de reconnaissance. Tout au plus on admet qu'une étape ultérieure de correction manuelle des résultats est parfois nécessaire, et

on minimise le problème en montrant que seule une faible proportion des documents sera concernée.

Nous prétendons que l'intransigeance quant à l'objectif d'automatisation ne sert pas les intérêts des utilisateurs, et peut conduire à des situations aberrantes. On observe en pratique que les erreurs commises par les outils d'analyse ont souvent un caractère répétitif. En impliquant l'utilisateur *durant* la reconnaissance, le système reçoit des informations en retour qui lui permettent de modifier son comportement en conséquence. Deux formes d'argumentation appuient l'intégration de l'utilisateur dans la session de reconnaissance :

- *Argumentation naïve.* On peut doter le système de fonctionnalités d'apprentissage incrémental servant à améliorer les connaissances du système à chaque intervention de l'utilisateur. On obtient un ajustement des paramètres avec une finesse qu'on ne peut pas espérer atteindre en prévoyant une phase initiale de configuration, où le système est entraîné sur une poignée d'échantillons.
- *Argumentation forte.* Les sessions de reconnaissance où collaborent l'homme et la machine sont fondamentalement plus riches que celles obtenues par un système non interactif, où les connaissances disponibles sont confinées dans les données reçues en entrée. Comme le soutient Wegner [204, 205], le comportement d'un programme interactif n'est pas réductible à l'exécution d'un algorithme automatique. Intuitivement, un algorithme qui exclut l'interaction perd une importante source de connaissances, comme un robot à qui on interdirait de consulter l'environnement perçu pour ajuster son itinéraire. De plus, un tel algorithme est incapable de tenir compte de l'écoulement du temps durant le calcul. Formellement, la différence porte sur le caractère fini (mais arbitrairement grand) ou infini du ruban dans une machine de Turing. Nous sommes portés à croire que l'argumentation de Wegner se manifeste en reconnaissance de documents, et qu'il y ait donc des cas où la minimisation de la main-d'oeuvre est inaccessible sans interactivité. Nous ne voyons malheureusement pas comment apporter une véritable preuve.

La manière conventionnelle de juger l'utilité des systèmes de reconnaissance est symptomatique du manque d'intérêt envers l'interface homme-machine. On évalue la qualité d'un outil de reconnaissance en cherchant à déterminer un taux de reconnaissance moyen. La mesure d'un taux d'erreur a l'avantage de faciliter la comparaison, mais présente aussi des inconvénients :

- la démarche est bien maîtrisée pour la reconnaissance d'objets élémentaires, mais les choses se compliquent pour les structures de documents;
- il est difficile de garantir que les conditions de test sont équivalentes;
- les résultats ne sont pas forcément transposables dans le contexte de différents utilisateurs;
- la valeur du taux d'erreur moyen ne renseigne aucunement sur les capacités d'*amélioration à l'usage*;
- mesurer le nombre d'erreurs n'est qu'un moyen indirect d'estimer la main d'oeuvre impliquée dans la reconnaissance.

Globalement, nous sommes persuadés que la coopération entre l'homme et la machine est la solution la plus raisonnable pour conduire une reconnaissance de documents à la fois efficace et fiable. En pratique, la qualité de l'environnement interactif sera finalement déterminante pour rentabiliser le travail du couple homme-machine.

2.2.2 Manque de souplesse

Les systèmes de reconnaissance actuels manquent de souplesse à plusieurs égards. En général, les systèmes complets sont dédiés à une application bien ciblée. Cela permet aux développeurs d'optimiser chaque partie du système, pour tenir compte des particularités d'utilisation. La démarche consiste à fournir un système «clé en main». Cela se justifie tout à fait pour les applications de grande envergure, comme le tri postal ou la lecture de chèques. Mais l'investissement est disproportionné pour les «petites» applications, qui demandent un prototype facilement reconfigurable.

Les prototypes paramétrés par une description de la classe de documents ont accompli un pas important vers des systèmes réutilisables. Toutefois, une reconfiguration passe par une description formelle, particulièrement pénible à élaborer à la main. De plus, une modification du modèle, même mineure, n'est pas possible en phase d'exploitation. Dans ces conditions, ces sytèmes de reconnaissance ne s'attaquent qu'à certaines applications, qui portent sur des grands volumes de données et où tous les documents sont supposés être composés avec rigueur.

Le fléau des systèmes de reconnaissance concerne moins la persistance d'un taux d'erreurs résiduel que le caractère répétitif des fautes commises. C'est en phase d'exploitation que les problèmes apparaissent.

Pour rendre plus crédible l'automatisation de la reconnaissance de documents, il faudrait que les résultats s'améliorent à l'usage. Malheureusement, les possibilités d'adapter les méthodes de reconnaissance pour supporter une forme d'apprentissage incrémental sont trop rarement exploitées. De plus, les systèmes qui offrent une interface graphique permettent la visualisation et la correction, mais l'interaction n'est pas mise au service de l'adaptation du moteur d'analyse.

La remarque sur la rigidité des analyseurs pris individuellement s'applique également au schéma d'exécution. En effet, il y a souvent plusieurs manières de combiner les analyses pour parvenir à une solution complète. Par exemple, le texte et la fonte sont des données interdépendantes, car la connaissance de l'une permet de guider la reconnaissance de l'autre. On peut donc reconnaître le texte puis la fonte, ou l'inverse, et les résultats ne seront pas toujours identiques. Aucune des alternatives n'est la meilleure dans l'absolu, puisque les résultats sont dépendants des particularités des données traitées.

Or, la plupart des systèmes actuels figent définitivement ces options stratégiques dans leur code. Souvent, le schéma d'exécution est conçu comme un enchaînement strict d'étapes, sans possibilité de retour en arrière. Trop peu de travaux remettent en question le schéma de base évoqué dans la figure 2.1. Nous proposons de corriger ce manque de souplesse sur le plan de l'architecture logicielle.

2.3 Révision de la problématique dans le projet *CIDRE*

Par rapport à l'état de l'art, le projet *CIDRE* est fondé sur une révision de toute la problématique en reconnaissance de documents. Cette section présente les quatre axes principaux de cette révision.

2.3.1 Reconnaissance assistée par ordinateur

Le projet *CIDRE* abandonne l'objectif de la reconnaissance automatique. Ce n'est pas une abdication, mais plutôt un changement dans la démarche. Nous sommes convaincus que les techniques de reconnaissance sont de nos jours rentables pour nombre de problèmes concrets, pour autant qu'un superviseur humain puisse être impliqué efficacement dans la session de travail.

Nous parlons de *reconnaissance assistée* pour mettre en évidence le rôle central de l'opérateur humain. Un environnement d'aide à la reconnaissance cherche à profiter au mieux de l'expertise humaine. Il ne suffit donc pas de développer une interface graphique au-dessus d'un système de reconnaissance automatique déjà existant. C'est toute la conception du système qui doit être imprégnée du souci de coopérer au mieux avec l'utilisateur. Fondamentalement, il faut revaloriser le rôle de l'opérateur humain pour le rendre actif dans la tâche de reconnaissance. Les fonctions d'analyse automatique sont offertes *pour qu'il puisse en profiter, non pour qu'il les subisse*. Nous adoptons ainsi une approche interactive de la résolution de problèmes [101, 206].

En reconnaissance des formes, on a l'habitude de réduire les données d'un problème à un *sous-ensemble d'apprentissage* et un *sous-ensemble de test*. Cette approche a l'avantage de fixer un cadre commun pour conduire les expériences. Toutefois, sélectionner au préalable des échantillons représentatifs se révèle très périlleux dans les applications réelles. Les expériences basées sur ce schéma débouchent sur des conclusions peu utiles en pratique. Avec notre approche assistée, nous mettons en évidence le caractère dynamique des connaissances accumulées par la machine.

L'objectif qui réconcilie l'approche automatique et notre approche interactive se formule de la manière suivante : le reconnaissance de documents doit être conçue de manière à *minimiser la main d'oeuvre* impliquée dans le processus (cf. section 9.1). *CIDRE* postule qu'on ne peut pas garantir cet optimum sans autoriser l'utilisateur à exercer une influence *durant* la session de reconnaissance, et non pas exclusivement dans une phase initiale de configuration, et une phase ultérieure de correction des résultats. Une fois que le système est bien entraîné, nous espérons que le nombre d'interventions de l'utilisateur tende vers zéro.

Notons que la revendication de systèmes centrés sur l'utilisateur a naturellement déjà été motivée [199], par exemple en ce qui concerne la recherche d'information [143], l'étiquetage logique [32], l'édition de texte [65] ou l'OCR [9].

Aussi banal que soit le souci de convivialité, le principe de reconnaissance assistée est à la base de tous nos travaux. L'étude aurait pu se concentrer sur les questions d'ergonomie. Mais les conséquences d'une véritable collaboration homme-machine surprennent par leur étendue : architecture logicielle, techniques d'apprentissage incrémental, nouvelles méthodes d'analyse, modélisation des documents, ou encore mesure des coûts de reconnaissance.

2.3.2 Réingénierie de documents au sens large

Le projet *CIDRE* ne spécifie pas la nature du problème de reconnaissance. Il n'est pas consacré à une classe de documents fétiche, ni même à une fonctionnalité déterminée. Dans un premier temps, nous cherchons à mettre en place une plateforme logicielle générale, susceptible de former la carcasse de différentes applications. Cela nous oblige à prendre du recul par rapport aux hypothèses sur le contexte d'utilisation.

On a l'habitude de définir le processus de reconnaissance de documents comme la reconstruction d'une structure conforme aux intentions de l'auteur, à partir d'une séquence d'images de pages. Cet énoncé conserve toute sa valeur, mais nous voulons rester ouverts à d'autres contextes d'utilisation, et anticiper de nouveaux besoins (cf. section 1.2.1). Ainsi, les données initiales sont peut-être disponibles sous une forme électronique comme PostScript ou HTML. La structure recherchée n'est pas forcément celle qui prévalait lors de la création du document. Une structuration partielle, portant sur certains champs uniquement, peut suffire. C'est pourquoi *CIDRE* se positionne dans la *réingénierie de documents* au sens large. Les applications évoquées dans la section 2.5 donnent un aperçu de la variété des besoins en réingénierie de documents.

2.3.3 Rôle de l'architecture logicielle

Le projet *CIDRE* postule que l'architecture logicielle est le principal facteur de réussite d'un système de reconnaissance. Nous ne dénigrons pas l'importance de l'interface graphique ou des techniques d'analyse, mais le fossé entre des composants isolés et un système complet est si grand que c'est dans la «colle» qu'il faut concentrer les efforts de conception. La raison tient à l'incertitude omniprésente : tous les résultats intermédiaires sont à considérer comme des hypothèses, et non des données exactes. La quasi-totalité des traitements impliqués dans la reconnaissance de documents ne seront jamais totalement fiables. On pourra continuellement en améliorer les performances dans un contexte précis ou inventer de nouveaux algorithmes, mais un certain taux d'échec semble inhérent au domaine.

La notion de chaîne de traitements est mal adaptée pour simuler l'expertise nécessaire pour conduire la reconnaissance de documents. Dans un environnement coopératif, c'est-à-dire interactif et adaptatif, le scénario d'analyse ne peut pas être prédit par le programmeur. Il faut plus de souplesse dans l'algorithme de contrôle, afin que le schéma d'exécution s'adapte en fonction (i) des particularités du document traité, et (ii) des interventions de l'utilisateur. L'architecture logicielle doit reposer sur des paradigmes qui encouragent la coopération homme-machine, mais aussi la coopération entre les différentes sources de connaissance du moteur d'analyse.

Plusieurs tentatives ont déjà été menées pour améliorer l'architecture logicielle. Par exemple, Chenevoy [44] ou Bayer [25] développent des approches basées sur les tableaux noirs. Hu [86] recourt à la logique floue pour contrôler l'évolution des traitements dans son prototype. De notre côté, nous désirons explorer les paradigmes multi-agents, car l'intelligence artificielle distribuée est une discipline qui affronte de face les problèmes d'autonomie, de localité, d'incertitude, de négociation, ou d'adaptation à l'environnement.

2.3.4 Modèles de documents

Un *document générique* ou *modèle de documents* est une description formelle des structures et de la présentation des documents à reconnaître. La plupart des travaux précédents [86] [44] considèrent le modèle de documents comme un paramètre d'entrée du système de reconnaissance. La motivation est double. D'une part, le programme peut être reconfiguré pour différents types de documents. D'autre part, le modèle n'est pas une information intrinsèque à un document, car la structuration dépend de l'usage qu'on veut en faire.

Le projet *CIDRE* abandonne l'hypothèse de l'existence préalable du modèle de documents, créé en dehors de l'environnement de reconnaissance. Lorsqu'une telle description n'est pas disponible, notre système doit être capable de le reconstituer de manière incrémentale durant la session de reconnaissance interactive. Les indications sur la classe de documents devraient être introduites par des exemples, et non à travers un langage formel.

En effet, la génération d'une description détaillée demande un tel effort que rares sont les situations où cela en vaut la peine. De plus, une description formelle ne tient pas compte des imperfections, voire des inconsistances auxquelles on doit faire face en pratique. Ce constat montre la difficulté d'utiliser un système paramétré par un modèle formel, mais ne remet pas en cause la notion de document générique : au contraire, celui-ci n'est plus considéré comme un fichier de configuration passif, mais plutôt comme une connaissance à enrichir lors de chaque reconnaissance. On pourrait même envisager de réutiliser ce modèle inféré pour *produire* de nouveaux documents conformes dans un éditeur structuré, comme Thot [173] par exemple.

L'acquisition incrémentale des modèles de documents soulève de nombreuses questions, en particulier sur la manière de représenter les informations. En effet, les formalismes usuels comme les grammaires sont mal adaptés, notamment par leur incapacité à véhiculer des informations statistiques. C'est pourquoi Rolf Brugger [38] a conçu un nouveau formalisme, basé sur les n-grams généralisés, capable de représenter les structures génériques. Les avantages de sa démarche sont multiples :

- le protocole d'interaction est très libre, car le modèle repose sur des motifs hiérarchiques locaux : la méthode se contente d'exemples incomplètement étiquetés, et peut fournir une interprétation sur une portion limitée d'un document;
- l'apprentissage incrémental devient trivial, puisqu'il se limite à la mise à jour de compteurs;
- l'algorithme de reconnaissance supporte les notions suivantes : confiance accordée à un sous-arbre, liste triée d'alternatives, révision locale, facteur de tolérance;
- la compatibilité avec les standards du génie documentaire est assurée grâce à une conversion depuis et vers SGML [37].

2.4 Amorces pour la conception d'un système assisté

Le développement d'un système de reconnaissance de documents assisté est une tâche ambitieuse, pour laquelle il n'y a aucune solution miracle. Cette section propose différents points de vue complémentaires qui peuvent servir à amorcer la conception d'une application. Chacune des optiques énoncées ici, et d'autres comme la modélisation orientée objet, mériteraient une étude approfondie. Cela dépasse le cadre du présent travail, qui se concentre sur les problèmes d'architecture logicielle.

2.4.1 Ergonomie ciblée sur une application

Le projet *CIDRE* insiste sur la nécessité de centrer l'environnement de reconnaissance autour de l'utilisateur final. Une base importante consiste à soigner l'analyse préalable des besoins de l'application. A l'aide d'entretiens avec les personnes les plus concernées, on essaye de construire un mode d'emploi complet de l'application. L'ergonomie de l'interface homme-machine est un élément crucial dans l'optimisation du temps passé à superviser les sessions de reconnaissance.

Si chaque application apportera son propre ensemble de fonctionnalités, on peut quand même dresser un catalogue général de sujets à discuter lors de l'élaboration d'une interface graphique :

- *Visualisation de l'image.* Il faut soigner le mécanisme de référence à l'image, puisque c'est la source d'information primordiale pour l'opérateur humain.
- *Temps de réponse.* Sachant que les techniques d'analyse d'images de documents peuvent être très gourmandes, les contraintes sur les temps de réponses risquent de remettre en cause certains choix.
- *Edition.* Dans certaines situations, l'utilisateur s'attendra à retrouver un mode d'interaction standard, par exemple lors de l'édition de texte ou dans un menu de sélection de fonte.
- *Interactions répétitives.* A chaque fois qu'on pressent des opérations répétitives dans le dialogue homme-machine, il faut réfléchir à des mécanismes pour automatiser la tâche. On sait par expérience que c'est notamment le cas lors de la correction d'OCR.
- *Interactions complexes.* Certaines opérations, comme la correction d'une mauvaise segmentation en lignes, sont particulièrement difficiles à conduire pour l'utilisateur; il est parfois possible de contourner ce problème en affinant les fonctionnalités, en l'occurrence en ajoutant des fonctions pour proposer automatiquement un ensemble raisonnable d'alternatives possibles.

Lorsque nous prétendons qu'il faut bien cibler une application pour concevoir le système de reconnaissance adéquat, cela ne veut pas dire que la classe de documents est alors définitivement fixée. Il s'agit de faire preuve de souplesse pour spécifier clairement l'application sans trop en limiter l'usage possible.

2.4.2 Panoplie d'analyseurs

En général, le développement d'une application complète nécessite la maîtrise de nombreuses techniques de reconnaissance. Fort heureusement, des progrès notoires ont lieu dans la diffusion des technologies en reconnaissance de document, et tout n'est pas à réinventer à chaque fois. Par exemple, on trouve sur les marché

Fonctionnalité abstraite	Description
reconnaître	interpréter une donnée
apprendre	s'adapter à une interprétation donnée (apprentissage supervisé)
estimer statistiquement	s'adapter à un échantillon typique, sans connaître l'interprétation (apprentissage non supervisé)
réviser	proposer d'autres interprétations possibles
pré-traiter	transformer une donnée pour l'améliorer
post-traiter	transformer un résultat pour l'améliorer
valider, critiquer	évaluer la confiance dans une hypothèse

Figure 2.2 : Fonctionnalités abstraites que peuvent offrir des analyseurs.

des OCR sous forme d'API (Application Programming Interface) [29]. Il existe même certains paquetages disponibles gratuitement [59, 89]. L'université de Washington met au point une collection d'analyseurs [128] adaptés au format DAFS [174, 62]. La littérature regorge d'algorithmes de pré-traitements d'images, de segmentation, de classification, etc.

Toutefois, avoir à disposition une collection d'outils d'analyse ne résout pas encore le problème de leur intégration en un logiciel cohérent. Une étape initiale dans ce sens consiste à dresser un inventaire détaillé des analyseurs. Voici quelques idées pour enrichir un tel inventaire, et établir une base pour la conception d'un système complet :

- faire ressortir ce que chaque paquetage peut offrir au niveau des différents domaines d'interprétations (segmentation, texte, fonte, structure logique, etc.);
- noter les outils selon leurs performances moyennes (rapidité et précision), par exemple en vue de recourir aux techniques «lourdes» seulement dans les cas douteux;
- définir des outils de plus haut niveau (cf. section 7.2) qui encapsulent le pilotage des analyseurs de base (p. ex. reconnaître la fonte puis le texte, ou organiser un vote entre plusieurs outils);
- caractériser les outils en les confrontant aux fonctionnalités abstraites évoquées dans la figure 2.2;
- définir une politique de gestion des résultats intermédiaires (par quels composants logiciels sont-ils accessibles, faut-il les conserver en mémoire ou les recalculer, etc.);
- définir une politique de paramétrisation (de quoi dépendent les paramètres, quelle partie du programme gère la configuration, etc.) ;
- spécifier l'origine et la destination des principaux flux d'information entre les sources de connaissances.

2.4.3 Allocation des ressources

La reconnaissance assistée de documents peut aussi être abordée comme un problème d'allocation de ressources. Le but est de minimiser les ressources humaines, en l'occurrence le temps total passé à superviser la reconnaissance. Les moyens s'expriment par le recours aux ressources informatiques, qui sont disponibles en quantités finies.

Cette optique met en évidence un déséquilibre entre deux perceptions du temps. Le temps de calcul se laisse combiner ou partager de manière rigoureuse. Il y a des règles précises pour optimiser un code [27], ou pour répartir la charge de calcul entre plusieurs CPU [190, 195]. D'un autre côté, l'efficacité du travail à l'écran résiste aux analyses naïves, car elle dépend du contexte. Ce n'est pas parce que l'opérateur atteint une vitesse de frappe de 300 car./min. qu'il sera capable de corriger des erreurs d'OCR éparses avec la même cadence.

Pour éviter que l'opérateur se trouve trop souvent en attente des résultats, il peut être judicieux de prévoir une phase préalable de traitements entièrement automatiques. Une partie des calculs sera alors déjà effectuée lorsque débute la session de travail interactive. Mais cela ne signifie pas que dès ce moment, on ne recourra plus aux analyseurs automatiques : dans la philosophie de *CIDRE*, l'environnement doit offrir une assistance continue, et ce n'est justement qu'à la suite des interventions humaines que les traitements les plus utiles peuvent être déterminés.

S'il est néfaste de bloquer le travail à l'écran par l'attente d'un résultat calculé, il faut se garder de noyer l'utilisateur en présentant tous les résultats trouvés de manière désordonnée. Le superviseur ne peut en

effet concentrer son attention que sur un sous-problème à la fois, et il a probablement envie d'enchaîner les opérations selon un certain ordre. Ainsi il est plus naturel de corriger l'OCR sur un paragraphe entier que sur des mots isolés de leur contexte. On pourrait alors concevoir un mécanisme pour traiter en priorité la *zone d'intérêt*, définie comme la partie des données où les résultats sont attendus dans les plus brefs délais. Ce serait particulièrement important dans un environnement où plusieurs analyses s'exécutent de manière concurrente : il faudrait traiter en priorité les requêtes qui servent les besoins les plus urgents de l'utilisateur.

Nos propositions en matière d'architecture logicielle tiennent compte des problèmes de gestion de ressources, puisque nous préconisons un support pour la programmation concurrente et distribuée.

Dans un environnement assisté, il arrivera que certains résultats soient jetés ou que d'autres soient évalués plusieurs fois au cours de la session. Nous estimons que cet éventuel gaspillage est un prix à payer pour obtenir un système vraiment convivial, et qu'à terme, seul le temps passé dans les interactions sera incompressible.

2.4.4 Métaphores

L'interdisciplinarité est une composante intrinsèque de la recherche en reconnaissance de documents, puisqu'elle nécessite des notions en reconnaissance des formes, en génie documentaire, en traitement d'images, en typographie, et bien d'autres encore. L'analogie avec d'autres disciplines restera une importante source d'inspiration pour les travaux futurs. Cette section évoque deux domaines qui sont comparables avec la reconnaissance de documents, puis énumère quelques principes à méditer avant de réaliser une application.

Reconnaissance de la parole Les problèmes en reconnaissance de la parole ont bien des points communs avec l'analyse d'images de documents. Notons par exemple les interdépendances entre divers niveaux d'analyse, tels que segmentation du signal acoustique en phonèmes, identification de mots, ou prosodie. L'intégration de multiples techniques dans un moteur d'analyse coopératif s'est révélée un défi essentiel depuis plusieurs années [47]. En outre, certaines techniques ont été explorées plus intensément en reconnaissance de parole que dans le monde du document. C'est le cas de l'usage de chaînes de Markov, même si l'application à la reconnaissance de caractères n'est pas récente [175].

Traduction automatique Dans le domaine de la traduction de langues naturelles, la complexité de la tâche est assez variable. Certaines phrases peuvent facilement être interprétées de manière automatique, à l'aide de dictionnaires et de règles simples. D'autres textes demandent une maîtrise parfaite des finesses des deux langues, et certaines expressions sont parfois presque intraduisibles. Par conséquent, une traduction parfaite totalement automatique relève de l'utopie. Une révision du rôle de l'opérateur humain dans un système de traduction assistée a été exposée dans un article de Kay en 1980 intitulé «*The Proper Place of Men and Machines in Language Translation*» [106]. Cette contribution, comparable avec l'approche *CIDRE*, a eu des conséquences majeures sur la conception des applications en traduction automatique. Une étude approfondie révélerait sûrement que certains enseignements sont transposables en reconnaissance de documents.

Principes directeurs Durant la phase de conception, certains principes informels peuvent servir à argumenter les différents choix. Voici quelques leitmotivs possibles, tirés du bon sens, et parfois contradictoires :

- L'homme n'est pas l'esclave de la machine. L'utilisateur doit pouvoir diriger la session de travail avec une certaine marge de liberté.
- Corriger, c'est réparer les erreurs déjà commises, et les éviter à l'avenir. Lors d'une transaction d'apprentissage, le système cherche essentiellement à modifier ses paramètres; il pourrait aussi automatiquement chercher d'autres occurrences de la même erreur sur les parties déjà analysées (cf. section 8.6).
- La première solution est souvent la bonne. Les techniques de *lazy evaluation* permettent d'attendre qu'une révision soit explicitement demandée, par exemple suite à un conflit entre plusieurs résultats.
- Il y a toujours quelque chose à vérifier. La perspective de sous-exploiter le CPU est assez dérangeante, quand on songe aux innombrables traitements qui peuvent être appliqués pour améliorer les résultats. Le terme d'*eager evaluation* se rapporte à une approche où on essaie de lancer le plus de traitements possibles, quitte à ce que certains résultats soient ensuite jugés inutiles.
- Le rapport qualité/prix est un bon critère de décision. On est souvent confronté à un compromis lorsque l'amélioration de la précision des analyses engendre une explosion des temps de calcul.
- Trop d'informations noie l'information. Ce principe s'applique à l'interface graphique, où on court le

risque de submerger l'opérateur humain en affichant un maximum d'informations, au lieu de cibler les éléments les plus pertinents.

2.5 Champs d'application pour *CIDRE*

Le projet *CIDRE* affronte des problèmes généraux en reconsidérant l'interactivité en reconnaissance de documents. Néanmoins, nous suivons de près l'évolution des besoins en la matière. Cette section donne un aperçu de quatre familles d'applications pour lesquelles des outils performants font encore défaut, et qui nous intéressent particulièrement.

2.5.1 Réédition de documents scannés

Une exploitation classique en reconnaissance de documents scannés consiste à en récupérer les résultats dans un éditeur. La plupart des OCR commerciaux sont déjà capables d'exporter le document dans un format d'édition. Toutefois, la notion de style n'est pas vraiment prise au sérieux, et les éditeurs visés, typiquement MS-Word, sont peu structurés.

Il manque encore un outil général d'aide à la ré-acquisition de documents, capable de retourner tout en amont de la chaîne de production avec un maximum de souplesse. Pour l'avenir du projet *CIDRE*, cette application est certes ambitieuse mais pas utopique, car certains outils sont déjà en place. Dans nos travaux [38] sur la reconnaissance de structures logiques, nous nous efforçons d'assurer la compatibilité avec les langages standard SGML [80] et DSSSL [93]. Cela traduit notre volonté de tirer profit des liens entre la reconnaissance de documents et la production. Notre recours à un support typographique (cf. section 7.3) constitue une autre tentative de jeter un pont vers le domaine éditorial.

Parmi les éditeurs de documents structurés qui se prêteraientt particulièrement bien à une extension vers la reconnaissance de documents, Thot [173] figure comme un autre candidat sérieux. Thot est un éditeur de texte complet, qui constitue une alternative aux logiciels LaTeX, MS Word, ou FrameMaker. En voici les principaux avantages :

- Les bases théoriques de Thot sont solides; on y traite par exemple soigneusement la distinction entre structure logique et présentation, ainsi que les références internes (p. ex. entre une note de bas de page et son renvoi) et externes (emprunt d'une partie d'un document à l'intérieur d'un autre).
- Thot dispose d'un excellent éditeur interactif WYSIWYG, qui est effectivement utilisé pour la production.
- Une API (Application Programming Interface) complète a été développée, et validée par de nombreuses réalisations, par exemple le navigateur Amaya.
- Il existe des convertisseurs vers LaTeX ou SGML.
- Le produit est maintenu par une large communauté de chercheurs.
- Tout l'environnement Thot, y compris les sources, est disponible gratuitement depuis le réseau, pour plusieurs systèmes d'exploitation.

Pour lier un système de reconnaissance à Thot, une première approche consiste à développer une passerelle qui traduise les résultats de reconnaissance, à savoir le modèle et l'échantillon reconnu, en schémas de structure et de présentation, respectivement en fichier d'édition Thot. Mais compte tenu de la richesse de l'environnement Thot, on peut même viser une intégration plus forte, en utilisant l'interface graphique Thot pour superviser les étapes de la reconnaissance. A notre connaissance, aucune équipe ne s'est encore penchée sur cette approche, mais elle nous semble très prometteuse.

2.5.2 Transformations de documents électroniques

A l'origine, la reconnaissance de documents était destinée à récupérer sous forme électronique des informations qui n'existeraient que sous forme papier. Dans le contexte actuel, les domaines d'application méritent d'être élargis.

La prolifération croissante des documents électroniques, loin de déprécier la recherche en analyse d'images de documents, amène au contraire de nouveaux besoins pour la restructuration. En effet, posséder une version électronique d'un document est différent d'en détenir une version exploitable. L'incompatibilité entre formats

de fichier est bien sûr un obstacle, mais en l'occurrence la standardisation ne résoudra que des problèmes de surface, puisque la structuration des documents dépend de l'utilisation qu'on veut en faire.

En pratique, on constate que les documents sont souvent échangés sous une forme non structurée. PostScript et PDF [2] se sont imposés comme des standards pour transmettre des documents à travers le réseau. D'une manière plus générale, la forme image peut être considérée comme une représentation commune des documents, quelle qu'en soit l'origine (cf. section 1.2.1). A chaque fois qu'une traduction directe entre deux structurations se révèle hors de portée, il reste donc toujours la possibilité d'effectuer un détour en générant des images, puis en recourant aux techniques d'analyses d'images de documents. Dans cette approche, la forme image est utilisée comme un *format pivot*. Le schéma de traitement devra être adapté pour exploiter au maximum les informations contenues dans la forme électronique, notamment les indications sur le texte et les fontes.

Dans le cas de la reconnaissance de documents PostScript, des solutions partielles existent déjà. Nous avons nous-mêmes effectué un premier pas en développant un convertisseur qui extrait les images, les caractères positionnés et les attributs typographiques pour les convertir vers le format DAFS (cf. section 8.7).

Pour prendre un autre exemple, de plus en plus de documents sont accessibles sur le réseau sous forme de pages WWW. En fait, la plupart des éditeurs, et bien d'autres applications encore, sont maintenant capables d'exporter vers le format HTML [144]. Mais la conversion réciproque est une tâche autrement ardue. En général, nous sentons un besoin croissant pour la restructuration des informations du Web. Il faut noter que les formats vont encore évoluer vers une meilleure structuration, notamment pour distinguer structures logiques et physiques. Les Cascading Style Sheets [130] (CSS) sont maintenant incorporés à HTML. Le langage XML [138] (eXtensible Markup Language) est en passe de devenir le prochain standard pour les échanges sur Internet. Il serait néanmoins déjà intéressant de développer, par exemple, une version interactive d'un outil `html2latex`.

2.5.3 Archivage et indexation

Si les applications précédentes sont consacrées à la transformation de documents utiles vers une forme exploitable, la recherche d'information constitue une autre facette de la gestion documentaire où des besoins croissants se font sentir. Tous les organismes sont amenés à conserver des documents dans le temps, ne serait-ce que pour des impératifs légaux. En raison de la diminution des coûts dans les supports informatiques, le stockage sous forme papier se voit concurrencé par l'acquisition des images via un scanner. Le problème majeur en archivage de documents, c'est la constitution d'index pour faciliter la recherche et l'accès dans la base de données. Nous pensons ici à l'archivage massif de documents hétérogènes multi-pages, pour lesquels il n'y a pas de clés d'accès naturelles, et où l'effort d'une reconnaissance idéale serait disproportionné. Pour donner une idée, supposons qu'on scanne en vrac tout ce qu'on trouve dans un local de travail; on obtiendrait une grande variété d'informations (documents complets, notes personnelles, aide-mémoires, directives de sécurité, etc.), particulièrement difficiles à organiser.

L'indexation par mots-clés a montré ses limites : elle se révèle fastidieuse, plus ou moins arbitraire (y a-t-il une autorité en matière d'adéquation des mots-clés choisis pour indexer?), et inappropriée aux éléments non textuels. De surcroît, elle n'aborde que superficiellement le problème de la navigation dans la base de données, qui est un complément indispensable aux requêtes de recherche d'information. L'indexation du texte entier (full-text indexing) n'est pas non plus une panacée; il faut entre autres tenir compte des erreurs d'OCR [197]. Surtout, l'indexation ne devrait pas se baser uniquement sur du texte, car les informations visuelles peuvent aussi aider à formuler des requêtes de recherche. On peut p. ex. restreindre la recherche à un certain format de papier (p. ex. A5), à une mise en page particulière (p. ex. 2 colonnes), voire rechercher des documents contenant des tableaux de chiffres, des logos, ou encore des diagrammes en camembert. Nous pensons que les techniques de reconnaissance structurelle peuvent aussi servir dans des applications d'indexation.

Dans ce contexte et pour valider l'approche *CIDRE*, nous sommes sur le point de concevoir un outil d'indexation, de recherche et de navigation assistées, basé sur les caractéristiques suivantes :

- reconnaissance partielle des structures de documents;
- recours à des descripteurs visuels;
- lancement d'analyses sur demande, dirigées en fonction des requêtes;
- transformation dynamique des documents dans la base;
- génération automatique de liens entre (parties de) documents;
- gestion duale entre contenu et forme image.

Dans cette application, la reconnaissance des documents n'est plus une fin en soi, mais un moyen d'en faciliter l'accès. On est moins strict quant à la forme du résultat, et surtout une reconnaissance complète n'est plus nécessaire, ce qui permet de réduire sensiblement les coûts d'indexation. Par ailleurs, nous envisageons certaines transformations sans quitter la forme image. Réduire les images pour présenter schématiquement la séquence des pages constitue un exemple simpliste d'aide à la navigation. Dans des problèmes de structuration plus intéressants, comme la construction automatique d'une table des matières, ou d'une liste des tableaux et figures, il peut être rentable de se passer de l'OCR, en utilisant astucieusement la segmentation et la reconnaissance de fontes. En fait, nos réflexions s'inscrivent dans une lignée de travaux récents qui montrent comment manipuler les images de texte à haut niveau, mais sans reconnaissance de caractères : Khoubyari et al. [107] détectent automatiquement les mots fonctionnels (articles, prépositions); Bagley et Kopec [11] proposent un éditeur d'images de texte; dans un contexte de recherche d'information, Trenkle et al. [202] identifient des mots entiers directement dans l'image; Chen et al. [41, 42] vont jusqu'à extraire des résumés d'articles.

Les premières réflexions nous ont amené à dépasser la frontière image-contenu en introduisant le concept de *résumé visuel* : l'idée revient à constituer, pour chaque document multi-pages à archiver, une représentation synthétique confinée dans une fenêtre à l'écran, qui sert (i) à offrir un aperçu visuel de l'information véhiculée, et (ii) à faciliter la navigation à l'intérieur de la base de données.

Nous définissons un *résumé visuel* comme une structure interactive qui présente, sous une forme graphique adaptée à la perception humaine, une collection de caractéristiques extraites des images d'un document. La modélisation détaillée de cette structure reste à définir, mais elle devra refléter des attributs dominants, comme le format de mise en page, la présence de schémas, de photos, de tableaux, de bibliographie, les mots les plus fréquents, les noms propres, les titres et légendes. L'interface utilisateur devrait non seulement faciliter la navigation entre le résumé visuel et des parties des documents, mais également la mise à jour de ce résumé, par exemple lorsque l'opérateur détecte qu'une information importante n'y figure pas. A terme, la spécification d'une requête de recherche s'articulerait autour d'esquisses ou de contraintes sur les résumés visuels. On peut aussi chercher à classer les documents, de manière à produire des résumés visuels selon une stratégie adaptée à chaque type de documents. Naturellement, ces connaissances devront s'acquérir par apprentissage.

2.5.4 Synthèse vocale de documents structurés

Bien que le terme multi-média soit très à la mode depuis quelques temps, il nous semble que peu de travaux font le lien entre images de documents et support sonore. Nous entrevoyons pourtant des applications très intéressantes :

- système de lecture automatique pour non-voyants[1];
- consultation de faxes à distance, par voie téléphonique;
- balisage audio de documents structurés [8, 96, 55].

Sur le plan architectural, la nouveauté réside dans l'intégration d'outils de synthèse vocale[2] dans le système de reconnaissance. Il ne suffit cependant pas de rediriger des résultats d'OCR vers un convertisseur acoustique [208, 158, 118]. Plusieurs sous-problèmes nécessiteraient une étude détaillée :

- Comment gérer la dimension temporelle? Bien plus que le support visuel, le son oblige à résoudre des questions de rythmes et d'ordonnancement lors de la présentation des résultats.
- Comment faire ressortir les structures de documents? L'auditeur devrait pouvoir percevoir les balises logiques, voire physiques, du document. On pourrait par exemple changer de «voix» pour les titres, ou annoncer les sauts de pages.
- Comment personnaliser le rendu acoustique? Le système devrait tenter de s'adapter aux préférences de l'utilisateur, par exemple pour la prononciation des noms propres, la vitesse d'élocution, ou le rendu de certaines abréviations.
- Comment récupérer les erreurs de reconnaissance? Pour les erreurs d'OCR, on peut offrir une commande pour épeler un mot bizarre. Pour les erreurs de chaînage de texte, il faudrait énoncer le début de chaque bloc candidat.

[1] Plus généralement, l'interaction homme machine pour personnes handicapées a fait l'objet de nombreux travaux [64, 137, 75, 203].

[2] Plus encore que pour l'OCR, plusieurs outils sont disponibles sur le réseau [88], par exemple `rsynth`.

- Comment parcourir le document structuré? Puisque nous manipulons des structures essentiellement hiérarchiques, on pense immédiatement à un parcours en profondeur. Toutefois, on devine l'intérêt d'une navigation plus souple, par exemple pour traiter les notes de bas de page ou les références bibliographiques.

Avec son approche assistée, le projet *CIDRE* offre un cadre adéquat pour aborder ces questions. En effet, l'interactivité prend encore davantage d'importance, en raison de la nature même du support sonore, qui impose une synchronisation forte entre l'homme et la machine. Quant au côté adaptatif du système, il s'étend au rendu acoustique, qui peut varier au gré des besoins ou des préférences de d'utilisateur.

Une problématique analogue se situe dans l'aide aux malvoyants, lorsque la lecture nécessite un fort grossissement de l'image. La solution courante se résume à l'emploi d'une loupe déplacée manuellement sur la page de document. Mais on oblige le lecteur à effectuer des sauts désagréables, notamment pour repérer le début de la prochaine ligne au milieu d'un paragraphe. Les techniques d'analyse d'image de documents permettraient d'améliorer le confort dans ce genre de situations, en dissimulant le formattage en lignes, colonnes, ou pages. Pour les documents essentiellement textuels, nous pensons à un système proche des téléscripteurs, avec un dispositif simple pour régler le sens et la vitesse de déroulement de la bande de texte. Ici, la dimension temporelle provient de contraintes imposées au support visuel, et non d'un recours au son. Nous nous intéressons beaucoup à ces applications de lecture assistée, d'autant que notre groupe de recherche vient de démarrer un nouveau projet sur la reconnaissance de pages de quotidiens.

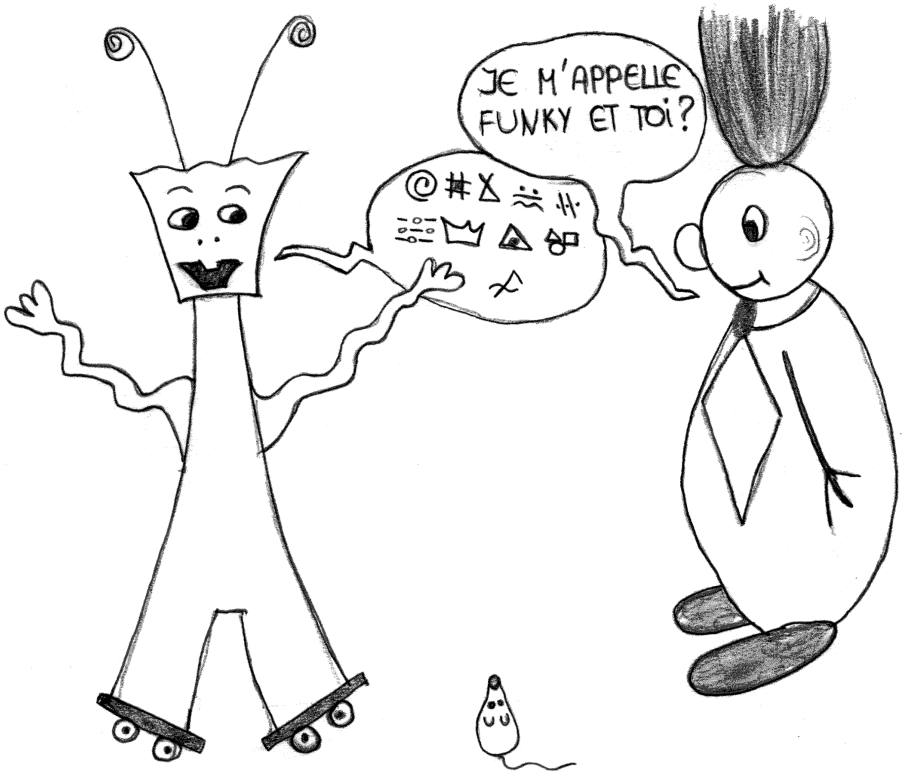
Conclusion

Certaines applications de reconnaissance de documents nécessitent des logiciels reconfigurables, qui manquent encore. Notre projet *CIDRE* s'attache à définir une architecture logicielle où l'homme et la machine coopèrent pour reconstituer la structure des documents de manière efficace et fiable.

Le prochain chapitre est consacré au dialogue homme-machine qui devrait s'établir dans un système de reconnaissance de documents assisté.

Chapitre 3

Dialogue homme-machine pour la reconnaissance de documents



Avant d'envisager le support de l'interactivité dans une architecture logicielle, il convient de s'interroger sur la nature du dialogue homme-machine dans un programme interactif de reconnaissance de documents. Ce chapitre analyse les caractéristiques des interactions homme-machine impliquées dans un processus de reconnaissance assistée.

Dans la première section, nous rappelons les concepts et les méthodes étudiés en interaction homme-machine. La notion de coopération homme-machine est développée dans la section 3.2. La section 3.3 explique les rôles respectifs de l'homme et de la machine dans un environnement de reconnaissance de documents. La section 3.4 présente trois styles de dialogue qui peuvent servir de base à l'élaboration d'un mode d'emploi. Dans la section 3.5, les sous-tâches de reconnaissance sont analysées séparément pour mettre en évidence les particularités des interactions requises. Enfin, la section 3.6 résume les incidences du dialogue homme-machine sur l'architecture logicielle.

3.1 Eléments théoriques en interaction homme-machine

L'innovation essentielle du projet *CIDRE* concerne l'interactivité et défend le concept de reconnaissance assistée. Il nous semble donc important de rappeler quelques éléments théoriques auxquels se réfèrent les professionnels en Interaction Homme-Machine (IHM). Au fil des années, l'IHM s'est constituée en discipline informatique à part entière. Cette section passe en revue les approches majeures qui façonnent la théorie des systèmes interactifs et le savoir-faire associé. Le but est surtout de montrer les potentialités de l'éclairage IHM dans la problématique *CIDRE*. Nous commençons par résumer la variété des sujets abordés en IHM, puis nous présentons brièvement quelques notions visant à modéliser le dialogue homme-machine. Le dernier point résume les contributions les plus intéressantes de l'IHM dans le contexte du projet *CIDRE*.

3.1.1 Survol du domaine

En guise de définition succincte, l'IHM étudie l'interaction entre les personnes, les ordinateurs, et les tâches. C'est une discipline très vaste, elle-même influencée par plusieurs branches d'études autres que l'informatique, telles que la psychologie cognitive, l'ergonomie, le génie logiciel, l'intelligence artificielle, la linguistique, etc. Les paragraphes suivants dressent un inventaire approximatif des thèmes, généralement interdépendants, abordés en IHM. Les mots-clés évoqués ici permettront de se repérer dans des ouvrages généralistes [98, 170, 50, 184, 83, 28]. La modélisation des interactions proprement dites constitue un thème central, et fait l'objet de la section suivante. Notons encore que l'IHM navigue souvent entre les aspects formels des interfaces utilisateurs (effort d'abstraction, modélisation rigoureuse), et les aspects empiriques (exemples, conseils pragmatiques).

Principes ergonomiques L'IHM propose toutes sortes de conseils sur l'ergonomie d'un système informatique : cohérence, concision, retour d'information, flexibilité, aide en ligne, etc. En marge, les travaux sur la programmation par démonstration [53] sont très intéressants dans le contexte de *CIDRE*, car ils montrent qu'on peut adapter le comportement d'un système à travers des interactions de haut niveau, en donnant des exemples de la tâche à accomplir.

Dispositifs d'entrée/sortie L'interface utilisateur est véhiculée par un ensemble de dispositifs d'entrée/sortie. Le triplet standard écran-clavier-souris est bien sûr largement discuté (p. ex. la loi de Fitt sur la difficulté à pointer un objet), mais l'IHM aborde aussi l'usage d'autres périphériques : synthèse et reconnaissance vocale, caméra, périphériques haptiques (liés aux gestes). Dans ce contexte, l'essor récent des domaines de la ralité virtuelle, de la vision ou de l'animation par ordinateur a constitué un puissant moteur d'innovation.

Environnement logiciel Le programmeur a aujourd'hui à disposition plusieurs outils logiciels qui facilitent la réalisation d'interfaces graphiques : panoplie d'éléments de dialogue (widgets), boîte à outils (p. ex. Motif [78]), langages de script (p. ex. Tk [165]) ou de programmation visuelle (p. ex. Labview [97]), générateur d'interface, etc.

Architecture logicielle L'organisation des sources d'un programme interactif est particulièrement difficile. Plusieurs modèles ont été proposés en IHM pour aider le programmeur à structurer son code, ou encore à le rendre indépendant du matériel ou du système de fenêtrage. Le modèle de Seeheim fut l'un des premiers à prôner p. ex. une séparation entre la présentation, le contrôle du dialogue et l'interface avec l'application. Parmi les modèles proposés utlérieurement, citons l'approche multi-agents PAC (Présentation-Abstraction-Contrôle) [51], ainsi que le système MVC (Model-View-Controller) utilisé en SmallTalk. Cet aspect de l'IHM a largement bénéficié de la programmation orientée objet.

Méthodologie de conception Plusieurs approches cherchent à guider le concepteur tout au long du cycle de vie du système : analyse des besoins, entretiens avec les utilisateurs finaux, spécification du produit, prototypage, évaluation, etc. La plupart de ces modèles sont proches du génie logiciel. IBIS (Issue-Based Information Systems) [170] est l'une de ces méthodologies, qui vise à gérer les décisions prises durant toute la phase de conception, et à dresser un schéma résumant les avantages et inconvénients de différentes options. Pour l'anecdote, on constate une propension à vouloir calquer le processus de conception sur une forme géométrique, comme une cascade, un V, un W, une spirale ou une étoile.

Contraintes temporelles Plusieurs essais de formalisation portent sur la dimension temporelle en IHM : taux d'affichage, temps de réponse, durée d'exécution d'une tâche, mémoire humaine à court et long terme,

etc.

Application Certains types d'application posent des défis à l'IHM, et contribuent au développement de principes plus généraux. C'est le cas du travail coopératif assisté (Computer Supported Cooperative Work, CSCW), de la réalité virtuelle, de l'aide à la conduite, de la recherche d'information (requêtes visuelles, navigation hypertexte), ou des jeux (notamment les simulateurs de vol).

Techniques de présentation Un savoir-faire complexe s'est développé sur la manière de présenter des informations à l'utilisateur : affichage synoptique, multi-fenêtrage, usage du son et de la couleur, messages d'erreurs, information visible ou observable, etc.

Comportement de l'utilisateur Modéliser l'activité cérébrale d'un utilisateur est un but louable mais hautement périlleux. Plusieurs tentatives ont vu le jour, comme le modèle du *processeur humain*. La tendance est aux modèles complexes, qui tiennent compte de la diversité des modes de raisonnement. Ces discussions touchent à la représentation des connaissance en IA classique, à la logique mathématique, à la biologie du cerveau, etc. D'autres travaux étudient les structures et les motifs invariants dans l'organisation des interactions entre personnes; ces approches sociales parlent d'actes du langage, de modèle de conversation ou de négociation.

3.1.2 Modélisation des interactions

L'un des objectifs majeurs de l'IHM vise à offrir un cadre de discussion pour spécifier les échanges entre le système informatique et l'opérateur humain. Cette section passe en revue les éléments qui touchent de près à la modélisation des interactions.

Style d'interaction On parle de *style d'interaction* pour évoquer les grandes catégories dans la manière d'interagir. Un style bien connu est basé sur les *langages de commande*. Un autre s'est développé autour d'une standardisation dans l'usage des *menus, boutons et formulaires*. Avec la *manipulation directe*, l'utilisateur agit de manière intuitive sur une représentation graphique du système réel (p. ex. éditeurs de texte WYSIWYG). Parmi les styles très évolués, citons le pilotage en *langue naturelle*. On parle de *mode* si les interactions autorisées par un programme prennent une sémantique différente suivant l'état courant du système.

Eléments de conception L'IHM propose toute une terminologie à l'intention du concepteur. Celui-ci cherche à schématiser de manière détaillée le système interactif dans ce qu'on appelle un *modèle de conception*. L'utilisateur se construit lui aussi, mais de manière plus floue, un *modèle mental* du programme. L'adéquation de ces deux représentations renseigne sur la qualité de la conception. On parle de *distances d'exécution/d'évaluation* pour désigner le décalage mental entre la formation d'un but et la perception des moyens (lancer des commandes, respectivement interpréter les réactions). Dans la description des intentions de l'utilisateurs, un *but* désigne un état final à atteindre (p. ex. avoir une lettre commerciale entièrement rédigée). Une *tâche* représente un ensemble structuré d'activités à entreprendre pour parvenir au but à l'aide de la machine (p. ex. éditer un paragraphe, mettre un mot en italique). Une *action* est atomique (p. ex. pointer, cliquer, frapper une touche). On nomme *analyse de tâches* l'étape de conception qui cherche à décrire l'organisation du travail, depuis le but jusqu'aux actions. En affinant la frontière homme-machine, le concepteur doit trouver un compromis entre les approches *descendantes* (des tâches aux fonctions) et *ascendantes* (des fonctions aux tâches).

Allocation de tâches L'expression *allocation de tâche* a beau être clairement positionnée dans toute la démarche de conception, elle cache en fait à elle seule tout un savoir-faire qui échappe aux méthodologies théoriques. La question abordée, capitale dans le cadre de *CIDRE*, est celle de la répartition des compétences entre l'homme et la machine. Les choix se révèlent spécialement ardus, car les alternatives sont souvent nombreuses, et l'évaluation de chacune nécessiterait un nouveau prototype. Preece souligne la difficulté de cette étape :

> «One of the most important decisions to be taken during the development of a human-computer system is to allocate tasks; to human, to computer or to human-computer system. [...] The task allocation stage [...] is certainly [...] one which will itself involve many iterations, prototyping of options, detailed analysis and user testing.» ([170] p. 441-443).

Formalismes utiles Plusieurs formalismes sont utilisés lors de la modélisation des interactions. Les *diagrammes de transition d'états* mettent en évidence les principaux états du système et les changements autorisés. Les *réseaux de Petri* permettent de se concentrer sur une représentation graphique des flux de données.

Acronyme	Nom complet
CCT	Cognitive Complexity Theory
KAT	Knowledge Analysis of Tasks
TAL	Task Action Language
TAG	Task Action Grammar
TKS	Task Knowledge Structures
YSS	Yoked State Space
ETIT	External Task Intertnal Task

Figure 3.1 : Quelques modèles proposés en IHM.

Des *grammaires formelles* sont parfois utiles pour spécifier par un langage la syntaxe des interactions permises. Toutes sortes d'*esquisses* servent à décrire le flux des données, schématiser les éléments de dialogue, ou représenter des scénarios typiques.

Quelques modèles Plusieurs auteurs ont proposé des modèles pour faciliter l'analyse des interactions. Le tableau 3.1 évoque les noms de quelques uns des formalismes existants [170]. Les avis divergent sur l'apport effectif d'une telle prolifération de modèles, mais ces théories font malgré tout partie de l'état de l'art en IHM. Voici une petit descriptif de trois d'entre eux (cf. p. ex. [50] pour les deux premiers, et [83] pour le troisième) :

- *GOMS (Goals-Operators-Methods-Selection rules)* part d'un principe de rationalité de l'utilisateur. Ses connaissances sont d'abord représentées par une hiérarchie de *buts* (états à atteindre). Dans ce modèle, un *opérateur* désigne une action élémentaire dont l'exécution provoque un changement d'état. Une *méthode* fait le lien entre but et opérateurs. Lorsque plusieurs méthodes conduisent au même but, des *règles de sélection* expriment les critères qui vont conditionner le choix présumé de l'utilisateur. Ce dernier aspect du modèle encourage la prédiction des performances, p. ex. en mesurant les temps d'exécution.
- *CLG (Command Language Grammar)* offre un cadre grammatical pour représenter en détail un système informatique sous quatre niveaux d'abstraction. Le niveau *tâche* organise hiérarchiquement les opérations conceptuelles (sans référence à la mise en oeuvre) que le système est supposé accomplir avec et pour l'utilisateur. Le niveau *sémantique* porte sur l'image du système perçue par l'utilisateur. Le niveau *syntaxique* spécifie un ensemble de commandes à travers leurs effets, leurs arguments ou leur contexte. Enfin, le niveau *interaction* correspond à une spécification lexicale qui décrit, en termes d'actions élémentaires, comment les commandes sont matérialisées. Le modèle CLG sert à la fois de guide de conception et de spécification informatique.
- *UAN (User Action Notation)* introduit une notation pour décrire les actions de l'utilisateur et les réactions de l'interface. Ce modèle permet une description très détaillée, destinée entre autres aux programmeurs. La notation, par ailleurs extensible, prévoit un ensemble de symboles pour exprimer les mouvements du curseur, les frappes au clavier, l'affichage de messages, la mise en évidence d'éléments graphiques, mais aussi la séquence d'événements, l'interruption, le choix, la concurrence ou l'attente. L'approche UAN a l'avantage de décrire de manière concise et rigoureuse le mode d'emploi d'un programme interactif.

3.1.3 Apports principaux au projet *CIDRE*

Démarche générale

Parmi les enseignements suggérés par la discipline d'IHM, certains points s'avèrent particulièrement importants dans le contexte du projet *CIDRE* :

- Il ne faut pas compter sur une définition «magique» de la coopération homme-machine. L'IHM édicte de nombreux critères de qualité, mais une étude du contexte de travail reste évidemment indispensable. C'est donc en nous plongeant dans le contexte de la reconnaissance de document que nous essayons dans la suite de ce chapitre d'exhiber des scénarios coopératifs.
- Toutes les méthodologies insistent sur la nécessité de séparer codage de l'interface graphique et noyau de l'application. Dans notre architecture, nous avons décidé de renforcer cette distinction par une frontière entre langages de programmation (cf. section 6.3).
- L'usage de notations formelles se justifie lorsqu'il s'agit d'exprimer la progression d'un programme

interactif, en fonction des interventions de l'utilisateur. C'est dans notre approche de l'évaluation des performances (cf. sect. 9.2) que nous avons senti l'intérêt de formaliser le comportement de l'utilisateur.

- L'innovation dans les paradigmes d'interactions concerne aussi des sujets moins tapageurs que la réalité virtuelle ou la reconnaissance de la parole. Le cas des *lentilles magiques* [31], discuté ci-après, montre comment enrichir l'interaction tout en restant dans le cadre usuel du multi-fenêtrage.

Face à la richesse de l'IHM, nous restons convaincus de l'intérêt d'une étude poussée sur l'interactivité impliquée dans un système de reconnaissance de documents. En revanche, nous n'avons pas cherché à respecter par la suite une des méthodologies décrites en IHM, pour au moins deux raisons. D'une part, le projet *CIDRE* n'est pas consacré à une application donnée, ce qui handicape sérieusement les discussions sur l'ergonomie. D'autre part, cette thèse adopte fondamentalement le point de vue de la reconnaissance de documents; il s'agit bien d'effleurer d'autres disciplines, mais pas de déplacer le centre d'intérêt de nos recherches. Cette remarque s'applique d'ailleurs aussi à nos emprunts à d'autres disciplines, comme les systèmes multi-agents ou la typographie.

Recours au concept de lentille magique

L'idée revient à généraliser la métaphore de la loupe. Une lentille magique [31] est une lucarne déplaçable qui modifie l'aspect de la surface sur laquelle elle est superposée.

Malgré la simplicité du concept, les lentilles magiques ouvrent de nouvelles perspectives en IHM. Suivant l'application, toute sorte d'effets se prêtent à une transformation visuelle : grossissement, simulation d'un écran noir-blanc, affichage des propriétés d'un objet, mise en relief de certains aspects de la structure, etc. L'idée de base conduit à de multiples extensions : combiner des lentilles entre elles, relier le déplacement de la lentille à un autre périphérique (deuxième souris), modifier non seulement l'aspect, mais aussi la manière d'agir sur les informations affichées sous une lentille, etc.

La notion de lentille magique s'accorde bien avec la reconnaissance de documents, où la superposition d'informations est un phénomène récurrent, avec les différentes interprétations que subissent les images de document. On pourrait par exemple mettre en oeuvre des lentilles pour l'OCR, la segmentation, la fonte, la structure logique, voire l'image originale si la vue normale en présente une version pré-traitée. Notons que l'utilité d'une lentille d'OCR serait suffisamment générale pour être à terme incorporée au système d'exploitation (au sens large, c.-à-d. y compris le système de fenêtrage). Cela permettrait de sélectionner du texte où qu'il se trouve à l'écran, p. ex. dans le titre d'une fenêtre, ou dans une image affichée dans un navigateur HTML. Des travaux récents [150, 131] sur des techniques d'OCR adéquates pour le monde WWW préparent le terrain à ce genre d'outils. L'annexe C présente notre solution pour simuler les lentilles magiques au sein d'une application écrite en Tcl-Tk.

3.2 Définition informelle de la coopération homme-machine

Il est souvent question de coopération dans les discussions sur l'interface homme-machine en reconnaissance assistée. Or ce concept est plus difficile à définir qu'il n'y paraît de prime abord. Nous allons aborder la notion de coopération homme-machine à travers ses aspects interactifs et adaptatifs. La discussion se veut pragmatique, car l'objectif reste la mise en oeuvre informatique d'un système de reconnaissance de documents, et non un débat théorique à caractère psychologique, éthologique ou autres.

3.2.1 Interactivité

On peut qualifier un programme d'interactif pour autant qu'il donne l'occasion à un utilisateur d'interagir durant une exécution. Toutefois, cette définition est bien trop succincte pour rendre compte de toutes les connotations du dialogue homme-machine. Afin de préciser le concept, nous allons discuter des propriétés qui caractérisent l'interaction prévue par un système.

- *Quantité.* La quantité des échanges homme-machine peut être estimée par plusieurs mesures, comme le nombre d'interactions minimal, moyen et maximal, en fonction des données à traiter. En fait, cet aspect comptable ne renseigne guère sur le rôle attribué à l'utilisateur.
- *Variété.* Le dialogue homme-machine est aussi conditionné par la diversité des interactions possibles. Un programme est d'autant plus interactif que le spectre des interventions de l'opérateur humain est large, puisque davantage de décisions sont alors soumises à l'influence humaine.

- *Initiative.* Ce ne sont pas seulement les caractéristiques intrinsèques des commandes offertes qui façonnent le dialogue. Il s'agit également de comprendre sous quelles contraintes elles sont autorisées. Les interactions sont parfois imposées par la machine sous forme de questions-réponses; dans d'autres cas, on limite certaines commandes à un contexte précis, par exemple à l'aide de modes. L'interactivité augmente si l'utilisateur est libre d'intervenir au moment et à l'endroit où il le juge opportun.

3.2.2 Adaptabilité

On parle de *système adaptatif* si le comportement apparent du système peut se modifier au cours du temps, afin d'améliorer les performances en fonction des besoins courants. On peut distinguer plusieurs formes d'adaptabilité, selon le moment où le système se modifie :

- *Phase de développement.* Les modifications nécessitent une retouche des sources du programme.
- *Phase de configuration.* Le logiciel lui-même reste invariable, mais il prévoit un mode spécial qui sert à en configurer les différents paramètres, afin de s'adapter à une nouvelle application cible.
- *Phase d'exploitation.* Dans le cours normal de l'exécution, et parallèlement à la fonction de base du programme, la paramétrisation change pour tenir compte des particularités des données traitées.

L'adaptabilité est aussi caractérisée par la manière dont elle est contrôlée, et deux tendances se profilent alors. Tout d'abord, les fonctions d'adaptation peuvent être *explicites*, au sens où le dialogue homme-machine leur réserve des interactions séparées du reste du travail d'exploitation. A l'extrême, l'interface utilisateur présente des tableaux de bord donnant accès à tous les paramètres internes du programme. Le système se laisse alors ajuster de manière très fine. Par contre, le dialogue devient dédié aux techniques d'analyse, et suppose que l'utilisateur ciblé possède une certaine expertise des algorithmes utilisés. En fait, cette approche place toute la responsabilité de l'adaptation sur l'opérateur humain, auquel le système ne fait qu'obéir.

A l'inverse, l'adaptation peut être menée de manière *transparente*. Dans ce cas, le système n'impose pas d'interaction spécialisée dans l'apprentissage, mais essaie de profiter de chaque intervention humaine pour «flairer» les possibilités d'amélioration. C'est une forme d'apprentissage évoluée, même si les techniques d'analyse peuvent être simples. En OCR par exemple, il n'est pas si difficile d'espionner chaque correction et de tenir à jour un catalogue d'exemples avec les images de mots qui ont posé des problèmes, afin de réduire les erreurs répétitives. Dans cette approche, l'utilisateur est déchargé du travail d'adaptation et se concentre sur la tâche d'exploitation. Toutefois, le dialogue engendre des effets de bord mal maîtrisés, qui n'aboutissent pas toujours à une paramétrisation idéale, au risque même de déstabiliser l'opérateur.

3.2.3 Coopération homme-machine dans *CIDRE*

Le terme de *coopération* dénote simplement une action conjointe, mais il a une forte connotation humaine et sociale, et c'est ce qui rend difficile une définition rigoureuse de la coopération homme-machine. Pour le projet *CIDRE*, nous adoptons la définition très approximative suivante : l'adjectif coopératif renseigne sur la *qualité des échanges homme-machine, au sens où l'interaction est enrichissante pour chaque partie.* On pourrait dire qu'il y a eu coopération homme-machine si l'utilisateur n'a pas de critique majeure à porter sur l'évolution d'une session de travail. Cela suppose que le système a répondu aux attentes de l'utilisateur, notamment lorsque celui-ci cherchait à améliorer les performances des opérations automatiques. Dans le contexte de *CIDRE*, les remarques de Schneiderman sur l'équilibre entre contrôle humain et automatisation permettent de mieux cerner notre notion d'environnement assisté :

> «Beyond performance of productive decision-making tasks and handling of failures, the role of the human operator will be to improve the design of the system. In complex systems, an opportunity always exists for improvement, so systems that lend themseves to refinement will evolve via continual incremental redesign by the operator» ([184] p. 86).

Dans un logiciel coopératif, le partage des compétences rend le dialogue profitable dans les deux sens. On vise un équilibre entre les aptitudes des deux participants, de façon à réduire les distances d'exécution et d'évaluation [50] (cf. section 3.1.2). Dans le pilotage d'un outil de segmentation par exemple, on aimerait éviter à l'utilisateur d'avoir à manipuler une foule de paramètres, ce qui nécessite une maîtrise des techniques d'analyse. Lorsque c'est possible, on peut proposer des options plus intuitives, telles que «segmenter plus finement».

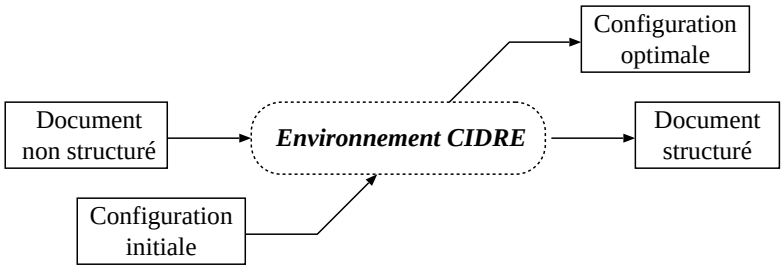


Figure 3.2 : Vision fonctionnelle de l'environnement de reconnaissance *CIDRE*.

L'utilisateur reste le maître de la session, mais dans le sens où il *peut* tout superviser, et non pas où il *doit* tout entreprendre. Le système automatique est considéré comme un élève, qui s'efforce de jouer le rôle d'un interlocuteur de haut niveau.

Le flou entourant notre approche de la coopération homme-machine traduit l'impossibilité de définir ce concept dans l'absolu. Pour cerner plus précisément cette notion, il faut donner des exemples concrets dans le cadre d'une application donnée. Dans ce sens, la suite du présent chapitre vise à esquisser certaines caractéristiques de la coopération homme-machine en reconnaissance de documents.

3.3 Rôles de l'homme et de la machine dans un système de reconnaissance

L'approche *CIDRE* accorde une importance capitale au dialogue homme-machine. Cette section tente d'esquisser les fonctions offertes par un système de reconnaissance, sans se confiner dans une application précise. Nous introduisons également les notions de stratégie et de scénario, qui serviront à structurer un peu les discussions autour de l'interface utilisateur.

3.3.1 Survol des fonctionnalités

Le projet *CIDRE* vise à faciliter la conception de systèmes de reconnaissance qui s'améliorent à l'usage. La figure 3.2 illustre comment cet objectif modifie la représentation fonctionnelle d'un environnement de reconnaissance, vu de l'extérieur. Une session de travail sert non seulement à transformer un document, par exemple de la forme image à une forme texte, mais également à adapter le système lui-même, à travers ses paramètres de configuration. A la fin d'une session, le document est sous une forme exploitable, et on espère en plus qu'un document similaire sera à l'avenir traité de manière plus efficace.

Vues de l'intérieur comme dans la figure 3.3, les différentes fonctionnalités de l'environnement de reconnaissance peuvent être positionnées suivant les deux axes automatique/manuel et reconnaissance/adaptation. Le moteur d'analyse se compose d'analyseurs de reconnaissance, comme un OCR, ainsi que d'outils d'adaptation, comme une routine d'apprentissage. De même, l'utilisateur intervient essentiellement pour éditer la solution courante, p. ex. en corrigeant du texte, ou pour affiner manuellement le comportement du système, p. ex. en ajustant des seuils de segmentation. Cela ne signifie pas que ces quatre catégories de fonctions se déroulent de manière indépendante. Voici quelques enchaînements possibles :

- Le système profite d'une édition manuelle pour ajouter la donnée éditée dans sa base de connaissances.
- Suite à une correction manuelle, le système lance un analyseur qui tente de trouver d'autres occurrences de l'erreur dans le document.
- Les résultats d'une analyse automatique sont si mauvais que le système demande à l'utilisateur de vérifier la paramétrisation.
- L'opérateur change un paramètre de configuration, et le programme relance l'analyseur concerné sur les données déjà traitées.

Le partage des compétences entre l'opérateur et le moteur d'analyse est discuté dans les prochaines sections 3.3.2 et 3.3.3. Le chapitre 7 est consacré aux méthodes d'analyse automatique. Les traitements à appliquer sur un document à reconnaître dépendent naturellement de l'application visée. Pour illustrer la variété des problèmes en reconnaissance de documents, voici une liste générale de sous-tâches à mettre en oeuvre :

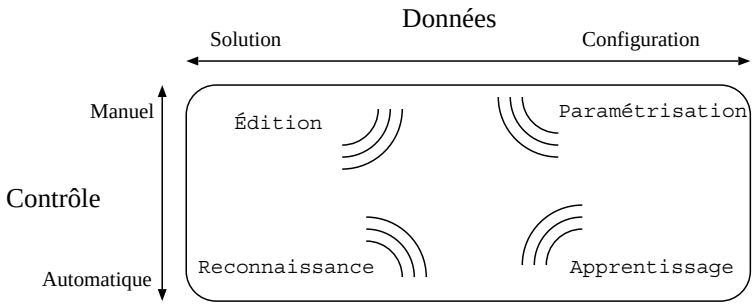


Figure 3.3 : Portée des opérations offertes par un environnement assisté.

- pré-traitements d'images;
- séparation en régions (en-tête, corps, pied de page, etc.);
- filtrage des figures et photos;
- segmentation en blocs, lignes et mots;
- reconnaissance de texte;
- identification de la fonte;
- décodage de la mise en page (mode d'alignement, marges, etc.);
- analyse de tableaux;
- construction d'entités logiques et étiquetage;
- chaînage des entités physiques et logiques (ordre de lecture);
- restitution des liens de références (vers les figures, etc.).

3.3.2 Stratégie d'analyse

Deux points de vue permettent d'appréhender le déroulement d'une reconnaissance assistée: pour le programmeur, la session de travail respecte l'enchaînement des opérations décrit par son programme. Mais en phase d'exploitation où le logiciel est invariable, ce sont les choix de l'opérateur humain qui déterminent le schéma d'exécution. Cette dualité est naïve mais importante pour discuter le partage des compétences dans les sessions de reconnaissance.

Nous appelons *stratégie* de reconnaissance le comportement prévu par le programme. Ce concept englobe à la fois les schémas de recours aux analyseurs automatiques, et les modalités du dialogue homme-machine. Pour un système entièrement automatique, établir une stratégie revient à décrire une chaîne de traitements plutôt rigide. Pour un système coopératif, la stratégie détermine le comportement à adopter en fonction des interventions de l'utilisateur. Par conséquent, la stratégie d'analyse fait le lien entre une collection d'analyseurs disponibles et les besoins précis de l'application. Or, on peut vouloir retarder certains choix jusqu'au moment de l'exécution.

Vu sous un autre angle, il s'agit de façonner la nature du travail humain en phase d'exploitation. Suivant la stratégie adoptée, on peut accorder différentes compétences au superviseur humain, qui vont de la simple correction d'erreurs au pilotage complet de tous les analyseurs. Ces décisions sont notamment influencées par la formation de l'utilisateur final, p. ex. selon qu'il est spécialiste en algorithmes de reconnaissance ou non.

En imposant des contraintes sur le déroulement des opérations, la stratégie décrit le «langage» des sessions de travail possibles. On y détermine l'influence théorique de l'utilisateur, par ses interventions directes, ou par l'apprentissage qu'il peut induire.

La figure 3.4 illustre le concept de stratégie d'analyse dans un problème de reconnaissance de texte. On suppose que le système possède les outils nécessaires pour reconnaître automatiquement la fonte ou le texte, ainsi que des commandes d'édition. Trois stratégies sont présentées sous forme de diagrammes. La stratégie (a) laisse la machine reconnaître la fonte puis le texte de toutes les lignes, avant d'autoriser l'utilisateur à

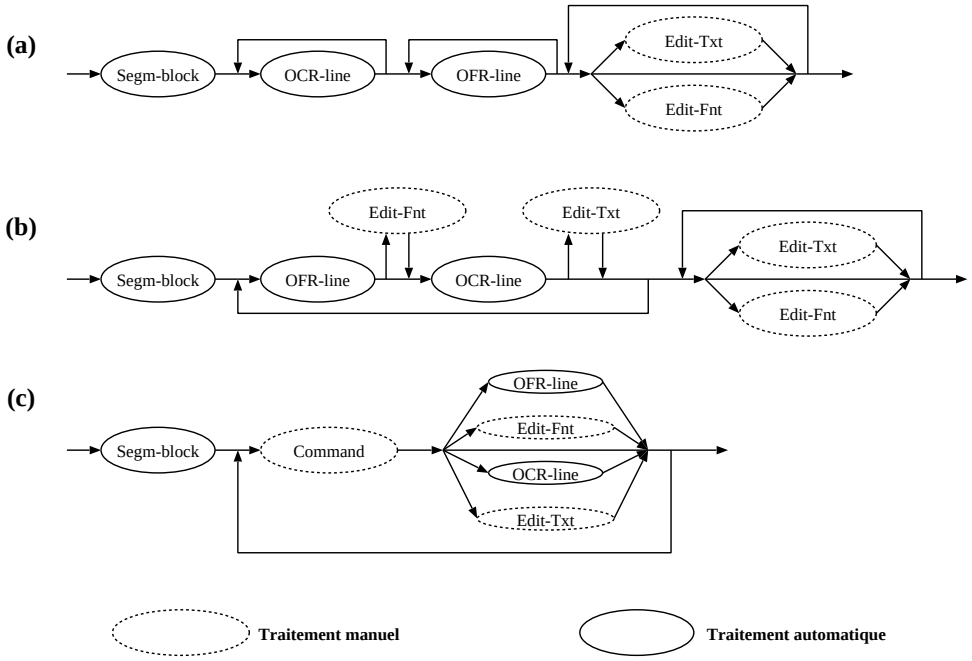


Figure 3.4 : Trois stratégies pour reconnaître le texte et la fonte.

éditer les résultats trouvés. La stratégie (b) prévoit de l'interaction ligne par ligne, après chaque analyse : l'utilisateur peut ainsi corriger la fonte avant que l'OCR ne soit appliqué. Avec la stratégie (c), l'utilisateur est libre de choisir à tout moment la prochaine opération manuelle ou automatique qu'il désire appliquer.

La stratégie détermine des caractéristiques importantes du système de reconnaissance, sur le plan de l'interactivité, de l'adaptatibilité, et de l'exploitation des résultats. Dans la stratégie (a), le système ne pourra pas profiter de la connaissance de la fonte pour guider l'OCR, et on ne peut pas bénéficier de l'apprentissage incrémental, puisque l'interaction est reléguée *après* toutes les analyses automatiques. Avec la stratégie (b), l'OCR peut consulter la fonte, et même lui accorder une certaine confiance puisque l'utilisateur a eu l'occasion de corriger ce résultat. La stratégie (c) ne garantit aucune source d'amélioration des résultats, mais n'en interdit aucune non plus : l'OCR pourra parfois se servir de l'information de la fonte, mais il y aura aussi des cas où c'est la reconnaissance des fontes qui pourra profiter du texte préalablement reconnu.

3.3.3 Scénario de reconnaissance

Une fois réalisé l'environnement de reconnaissance, c'est à l'utilisateur d'influencer le cours d'une session de travail. Nous parlons de *scénario* de reconnaissance pour désigner un enchaînement précis d'opérations dans une session particulière. On peut le voir comme une trace des événements survenus durant la reconnaissance d'un document.

Un scénario se réalise toujours dans le contexte d'une certaine stratégie, dont il forme un représentant particulier. Mais deux stratégies différentes permettent parfois d'engendrer un scénario identique. Le scénario décrit l'influence effective de l'utilisateur dans une situation concrète.

En général, plusieurs scénarios sont susceptibles d'aboutir au même résultat, pour un document et un environnement donnés. L'analyse de la rentabilité des scénarios (cf. chap. 9) peut alors servir à conseiller l'utilisateur dans ses choix, voire à définir ensuite une stratégie plus contraignante qui évitera les pires scénarios.

La figure 3.5 illustre le concept de scénario de reconnaissance dans le cas de l'analyse d'un bloc de texte contenant deux lignes. Le scénario (a) décrit une séquence d'opérations automatiques puis manuelles qui permettent de reconnaître correctement le bloc. Le scénario (b) aboutit au même résultat final, mais à travers une séquence différente d'opérations, qui aurait pu être généré par l'une des deux dernières stratégies de la figure 3.4. Dans cet exemple, le scénario (b) semble plus efficace que le scénario (a), puisque l'utilisateur a effectué une correction de moins.

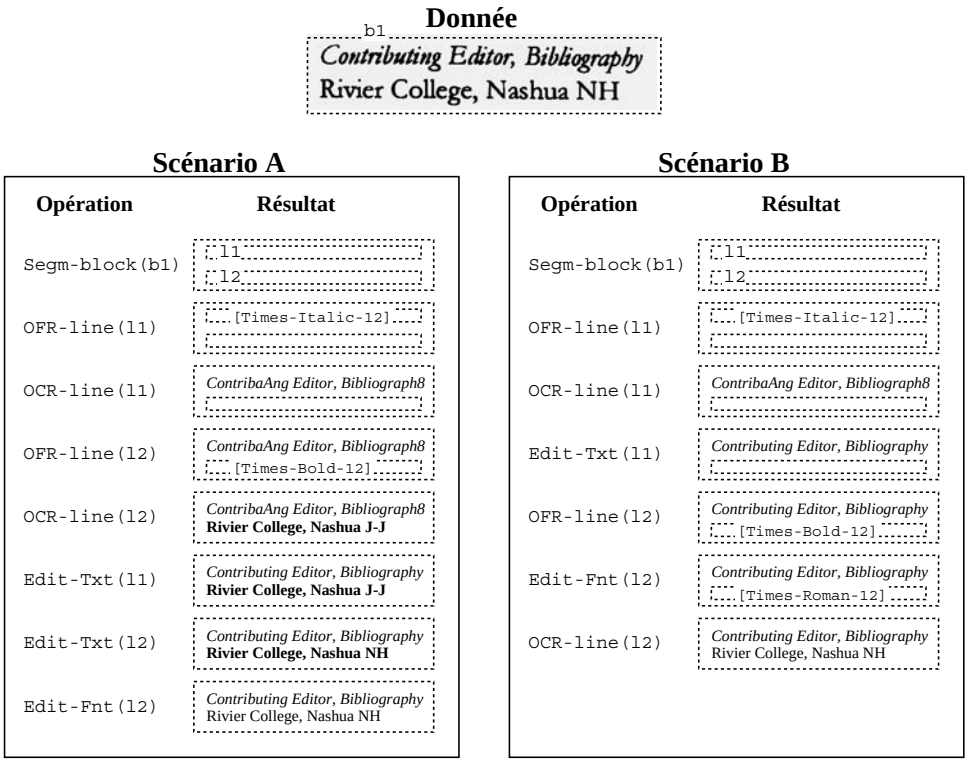


Figure 3.5 : Deux scénarios pour reconnaître un échantillon.

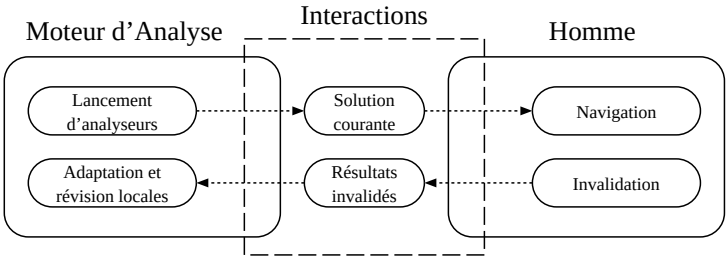


Figure 3.6 : Interaction basée sur l'invalidation.

3.4 Styles de dialogue adaptés à *CIDRE*

Afin d'illustrer la variété des choix qui s'offrent au concepteur de la partie interactive d'un système de reconnaissance de documents, nous allons esquisser trois styles de dialogues homme-machine qui peuvent servir de base au cahier des charges d'un environnement assisté. Notons que nos trois propositions ne sont probablement pas à adopter à la lettre sous leur forme extrême, mais qu'elles apportent néanmoins des éléments de réflexion enrichissants.

3.4.1 Environnement d'invalidation

Ce principe d'interaction s'énonce plutôt simplement, mais cache un moteur d'analyse complexe. Les résultats d'analyse sont ajoutés au fur et à mesure dans la solution courante, qui est affichée en permanence. Le rôle de l'utilisateur se limite à *invalider* des parties de cette solution, sans aucune contrainte. C'est au système de reconnaissance automatique de réagir correctement, en proposant une nouvelle interprétation, en remettant en cause d'autres résultats qui dépendaient de la donnée erronée, ou en accumulant des connaissances par apprentissage.

Nous estimons que ce style de dialogue devrait être mis en oeuvre de manière totalement asynchrone, afin de permettre à l'utilisateur d'intervenir à n'importe quel moment de la session de travail. Une solution alternative serait de soumettre chaque résultat à l'invalidation de manière dirigiste, c.-à-d. dans un ordre

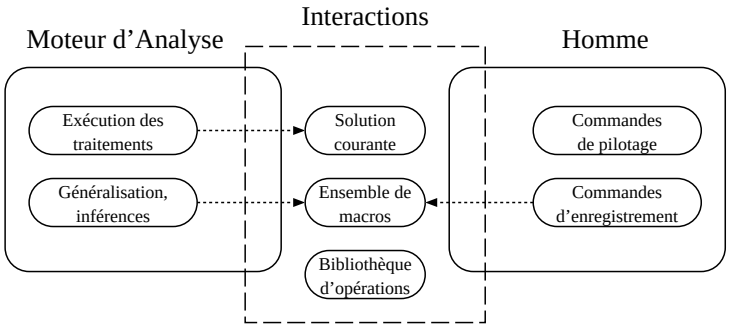


Figure 3.7 : Interaction basée sur des macros.

choisi par la machine.

Avec cette approche, esquissée sur la figure 3.6, les interventions de l'utilisateur sont extrêmement simples, puisqu'elles se résument à la désignation d'un résultat défectueux. Il suffit donc de soigner la présentation de la solution en cours, afin de faciliter la détection des résultats erronés. En fait, la réussite d'un tel environnement dépendra directement de la puissance du moteur d'analyse. Il faudrait notamment veiller à ce que l'utilisateur ne perde pas son temps à invalider les interprétations successives d'une même donnée.

3.4.2 Enregistrement de macros-fonctions

La figure 3.7 schématise une interface homme-machine où c'est l'utilisateur qui commande les différents traitements à appliquer sur les entités. Mais pour lui faciliter la tâche, le système offre un mécanisme pour enregistrer une séquence d'opérations sous forme de *macros-fonctions*. Ces nouvelles fonctions s'ajoutent ensuite aux routines de base prédéfinies qui interfacent les analyseurs intégrés au système. A mesure que la session avance, l'utilisateur se construit donc des fonctionnalités de plus haut niveau adaptées au document traité. Il peut éventuellement lancer ses nouvelles commandes en arrière-plan, de façon à pouvoir continuer à travailler de manière asynchrone. A l'extrême et dans le cas idéal, l'utilisateur finit par déterminer lui-même la macro qui va conduire toute la reconnaissance du document de façon automatique, pour une certaine classe de documents.

Cette approche implique la définition d'un protocole pour programmer les nouvelles fonctions, par exemple à l'aide d'un langage de script. Une partie du travail de programmation est ainsi retardée jusqu'à l'exécution, parce qu'on revendique l'hypothèse qu'aucune combinaison d'analyseurs n'est optimale pour tous les types de documents. Par ailleurs, l'ajout d'un nouvel analyseur dans le système ne demanderait en principe que l'effort d'en assurer sa compatibilité en tant que commande.

Si la notion de macro-fonction est bien connue de certaines applications (p. ex. les tableurs), elle continue néanmoins à poser des défis par rapport à la programmation en général. L'objectif central vise à simplifier la programmation, et le moyen consiste à exploiter au maximum l'interaction homme-machine, au lieu d'utiliser directement un langage de programmation. Dans ce contexte, on peut se référer aux récents travaux sur la *programmation par démonstration* [53].

L'idée des macros s'applique aussi à des manipulations d'édition. Par exemple, elle pourrait servir dans la délimitation des régions dans une page, lorsque les positions sont invariantes entre les pages. L'application d'une même correction d'OCR sur plusieurs occurrences (cf. section 8.6) peut aussi être considérée comme une forme de macro.

3.4.3 Collaboration faiblement couplée

Plutôt que de répartir a priori les tâches entre l'homme et la machine et de prévoir des protocoles de dialogue, on peut aussi adopter le point de vue suivant, schématisé sur la figure 3.8 : puisque chaque étape de reconnaissance peut s'accomplir manuellement ou automatiquement, laissons l'homme et la machine travailler de manière concurrente, sans spécifier d'échange explicite entre les deux parties. Comme avec la stratégie d'invalidation, les interventions de l'utilisateur sont définies de façon asynchrone.

Avec cette approche faiblement couplée, les rôles de l'homme et de la machine sont symétriques. A la base, il s'agit pour chacun de choisir des commandes pour parvenir au résultat final. Lors des étapes intermédiaires, le principe consiste à mettre à disposition de l'autre partie les informations sur les décisions et les résultats

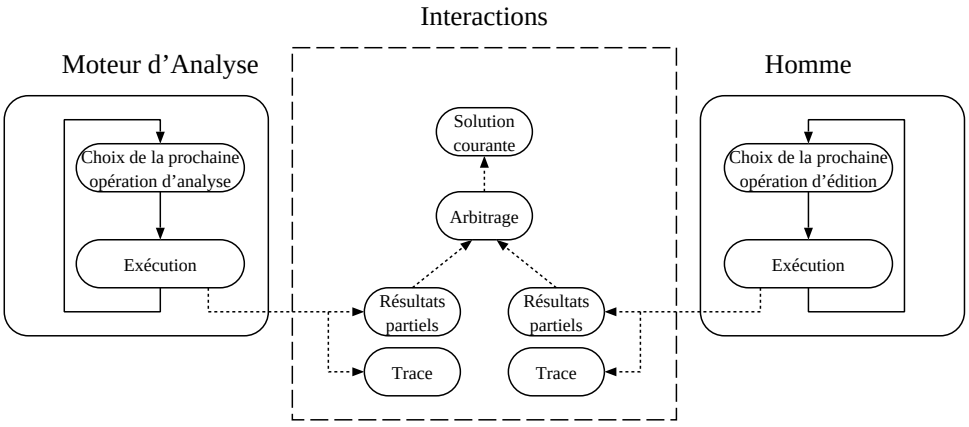


Figure 3.8 : Interaction basée sur un couplage faible.

obtenus. La lecture de ces données peut alors être prise en compte pour influencer la suite des opérations. Le dernier élément de cette stratégie est une procédure d'arbitrage pour lever les conflits entre résultats manuels et automatiques. On peut par exemple considérer les interprétations humaines comme autant de contraintes imposées au moteur d'analyse.

Ce style de dialogue comporte des aspects positifs et négatifs. Le mode d'emploi sera simple, car il se limite aux commandes d'édition. Le système peut entièrement se passer de l'utilisateur : une fois bien rodé, il peut fonctionner comme un programme batch. Mais tout le problème de l'amélioration des performances est relégué au niveau de l'algorithmique, car le moteur d'analyse ne peut que deviner les adaptations nécessaires sur la seule base des traces d'édition. Par ailleurs, rien ne garantit que les étapes choisies par le système correspondent aux attentes de l'utilisateur. Par exemple, il faudrait éviter que l'utilisateur se concentre sur la troisième page, alors que la machine est occupé à reconnaître la deuxième.

3.5 Interactions dans les sous-tâches de reconnaissance

Dans l'optique d'un mode d'emploi complet, on peut commencer par imaginer les formes d'interactions propres à chaque étape de reconnaissance. Cette section passe en revue les sous-tâches principales d'une reconnaissance de document, et envisage les propriétés d'une interface utilisateur centrée sur les besoins.

Il est clair que certaines applications impliquent des tâches plus spécifiques (p. ex. reconnaissance de tableaux ou de formules mathématiques). A l'inverse, d'autres tâches impliquées en reconnaissance de documents ont une portée bien plus générale : c'est le cas notamment des opérations de traitement d'image, destinées à réduire les nuisances qui handicapent les analyseurs automatiques (biais, bordures noires, taches, effets de discrétisation, etc.). En l'occurrence, les environnements de retouche d'image comme Photoshop donnent l'exemple sur la manière d'organiser l'espace de travail.

Le but de cette discussion est de donner des idées sur la conduite interactive de quelques sous-tâches prises isolément, comme segmenter une page, reconnaître du texte, ou identifier les fontes. C'est pourquoi nous ne faisons pas référence à un style de dialogue donné (cf. section 3.4), ni aux interdépendances entre niveaux d'analyse (cf. section 7.2); ces paramètres relèvent de l'organisation du travail dans la globalité du système. Par ailleurs, certaines caractéristiques sont communes à la plupart des tâches de reconnaissance. Ainsi, quel que soit le niveau d'analyse, l'utilisateur organise son travail selon une combinaison de deux cycles dominants :

- *Cycle de reconnaissance :* dans cette dynamique, l'utilisateur concentre ses efforts sur la solution finale. En guise de moyens, le système lui offre des commandes d'édition manuelle et des analyseurs. Lancer une analyse automatique revient à indiquer, explicitement ou implicitement, (i) la nature des entités à reconnaître, (ii) la portion du document où doit porter la recherche, et (iii) la configuration des analyseurs.

- *Cycle d'apprentissage :* dans cette dynamique, l'utilisateur cherche à adapter le système, afin d'en améliorer les performances. Le schéma est grossièrement le suivant : (i) détecter un problème (p. ex. constater une erreur systématique); (ii) apporter les connaissances nécessaires (p. ex. corriger manuellement quelques occurrences); (iii) s'assurer que le système a bien appris (p. ex. en lançant une

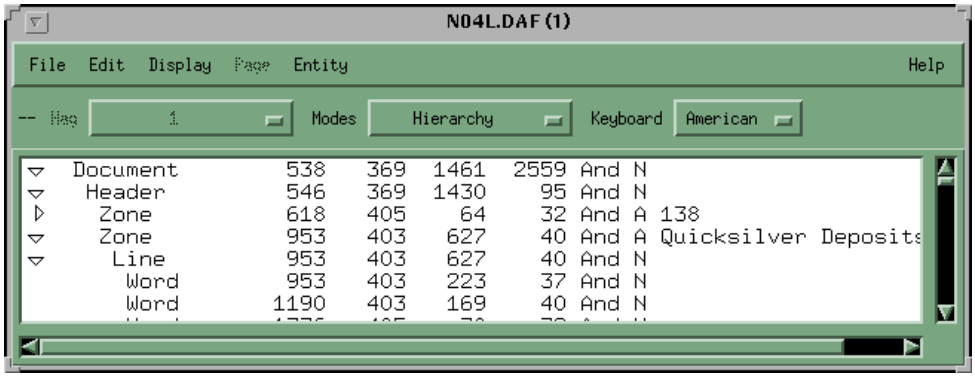


Figure 3.9 : Vue hiérarchique dans Illuminator.

nouvelle analyse).

3.5.1 Segmentation assistée

Nature de la tâche La segmentation est l'étape qui découpe les pages en entités physiques, de manière hiérarchique. L'analyse est guidée par des contraintes sur les séparateurs (seuillage sur les espaces blancs, homogénéité des interlignes) ou sur les entités elles-mêmes (dimensions bornées par un intervalle selon le type d'entité).

Présentation On représente généralement les résultats en dessinant les enveloppes rectangulaires autour des entités reconnues (lignes, blocs) de la page. On peut aussi proposer une vue qui met en évidence l'arborescence, comme dans Illuminator, l'éditeur de documents DAFS [174] (cf. figure 3.9). Pour faciliter la détection d'erreurs, le système pourrait mettre en évidence les frontières douteuses, p. ex. lorsque l'espacement est très proche du seuil. Pour ne pas surcharger l'affichage, on peut recourir à une *lentille magique* (cf. section 3.1.3 et annexe C) pour présenter les enveloppes, surtout pour les petites entités comme les mots.

Commandes d'édition La construction d'une entité physique s'effectue soit par détourage manuel (p. ex. à la souris), soit de manière ascendante en sélectionnant la séquence d'entités physiques correspondant à la découpe. L'aspect récursif d'un résultat de segmentation encourage deux opérations d'effacement distinctes, selon que les composants sont eux aussi détruits ou non. Notons que la délimitation manuelle d'une frontière exacte est une tâche ardue. Nous conseillons donc de la faciliter, soit en autorisant une frontière approximative, soit en limitant le choix aux frontières les plus raisonnables.

Commandes d'analyse L'ajustement manuel des paramètres de segmentation est une tâche pénible. Une manière de contourner cet obstacle consiste dans l'estimation automatique de paramètres, comme le fait Azokly [10]. Le système peut aussi offrir deux commandes qui relancent l'analyse suite à une sous- ou sur-segmentation.

3.5.2 Reconnaissance de fontes assistée

Nature de la tâche La tâche de reconnaissance des fontes est conditionnée par deux facteurs essentiels, dont l'interface graphique doit tenir compte : (i) l'ensemble des fontes candidates possibles, et (ii) les contraintes d'homogénéité, qui indiquent dans quelles conditions les changements de fonte sont le plus probables. Sans aucune contrainte d'homogénéité, la tâche consisterait à associer une fonte à chaque caractère, ce qui est extrêmement périlleux. Si on articule le dialogue autour de ces deux arguments, les opérations principales sont : (i) la sélection d'un sous-ensemble de polices et de styles, et (ii) la délimitation d'une portion d'image où le texte a une fonte homogène.

Présentation L'attribut fonte se manifeste naturellement par le dessin des caractères reconnus. Nous supposons que le système est capable d'afficher du texte en respectant les fontes reconnues (cf. section 7.3.2). Nous préconisons une vue où les résultats se superposent à la forme image [91], pour permettre à l'oeil humain de repérer les incohérences. L'interface doit aussi offrir un moyen de visualiser le nom précis de la fonte associée à une entité, notamment lorsque la fonte est trouvée avant l'interprétation textuelle. Pour gérer ces informations superposables, le concept de *lentille magique* offre une solution tout à fait conviviale.

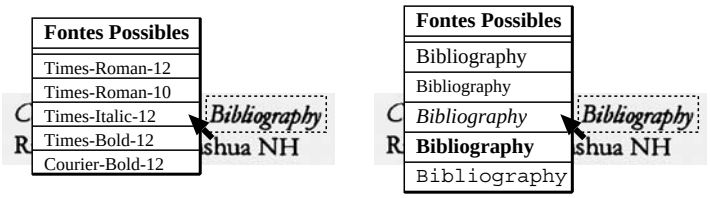


Figure 3.10 : Deux types de menu pour le choix de la fonte.

Figure 3.11 : Affichage du texte sur les images de page avec Illuminator.

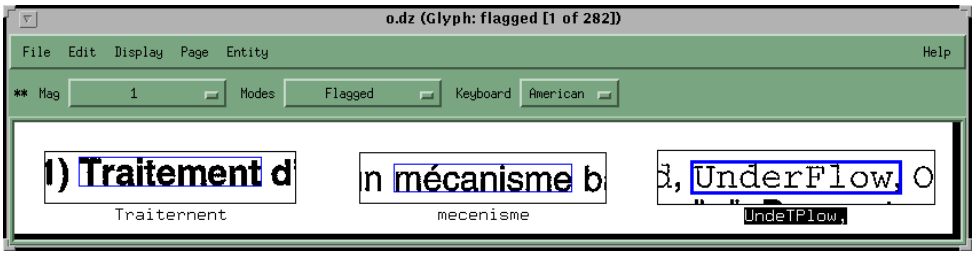


Figure 3.12 : Vues limitée aux mots douteux avec Illuminator.

Commandes d'édition Le choix des fontes est une opération usuelle en production de documents. Elle s'effectue généralement à travers des menus qui énumèrent respectivement l'ensemble des fontes installées, les tailles les plus fréquentes, et les différents styles prévus (italique, gras). En phase de reconnaissance, on pourrait restreindre le choix et ne proposer que les fontes raisonnables, en se basant sur l'image du texte sélectionné. De plus, la décision serait parfois facilitée si on présentait les candidats directement par l'aspect que prendrait l'élément textuel, comme sur la droite de la figure 3.10. Quant à la délimitation des portions du document, l'environnement devrait prévoir la sélection d'un ensemble d'entités physiques ou logiques, en s'inspirant de ce qu'on trouve dans les éditeurs. Lors de la validation, il peut être utile de mettre en évidence toutes les entités qui ont une fonte commune.

Commandes d'analyse Au moment de commencer la reconnaissance d'un nouveau type de documents, l'utilisateur doit d'abord déterminer l'ensemble des fontes utilisées. C'est une étape difficile, même pour un typographe professionnel. Nous proposons un premier utilitaire pour reconnaître, sur une petite entité (un mot), les fontes les plus proches parmi *toutes* les polices installées. Un tel outil servirait à guider l'utilisateur pour déterminer un ensemble raisonnable des fontes présentes, notamment en cas de doute. En cours d'étiquetage typographique, ce sont les changements de fonte dans le corps du texte (p. ex. un mot en italique) qui sont difficiles à détecter. Pour aider à valider les résultats, il faudrait un outil qui mette en évidence les mots (ou les caractères?) qui semblent en désaccord avec la fonte choisie pour les blocs. De cette manière, après avoir associé une fonte à tout un bloc, le système pourrait faire ressortir les quelques mots qui sont formattés autrement que le corps du texte.

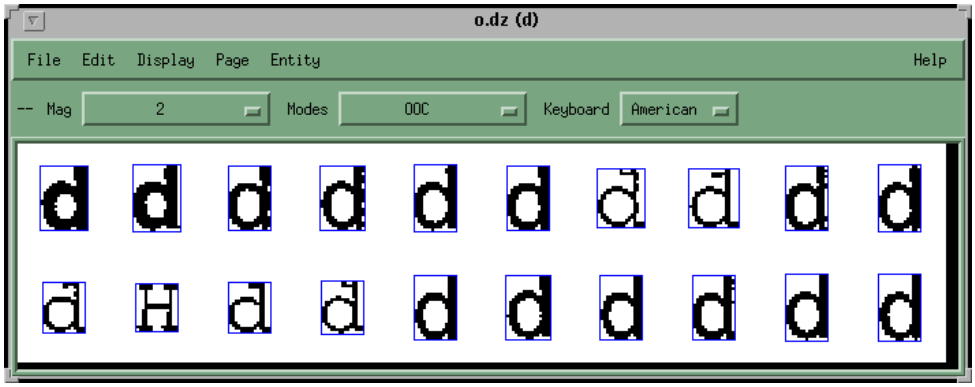


Figure 3.13 : Groupement de caractères hors contexte avec Illuminator.

3.5.3 OCR assisté

Nature de la tâche La reconnaissance de caractères (OCR) a été le sujet le plus étudié de toute la reconnaissance de documents. Considérons ici qu'elle consiste à attacher une chaîne de caractères à l'enveloppe d'un mot. Plusieurs paramètres peuvent servir à guider cette opération : alphabet restreint (p. ex. aux chiffres), contraintes sur la fonte, respect d'un lexique, ou conformité à une langue naturelle.

Présentation Nous avons déjà évoqué l'idée d'une vue où le texte reconnu est superposé à l'image de la page. Comme le montre la figure 3.11, c'est un mode supporté par Illuminator, l'éditeur de documents DAFS [174]. Une amélioration consiste à présenter le texte en respectant l'alignement horizontal, et avec les fontes correctes. Pour mettre en évidence les mots douteux, on pourrait recourir à un changement de couleur. Dans le même esprit, Illuminator propose une vue qui résume tous les résultats douteux, comme sur la figure 3.12. Ce logiciel définit aussi le mode de la figure 3.13, qui présente pour chaque caractère un regroupement de toutes les occurences présentes dans le document. Ce mode permet aussi de repérer les erreurs, telles que le 'H' reconnu comme un 'd' dans l'exemple de la figure 3.13. Il faudrait peut-être affiner la fonctionnalité, pour présenter ensemble les occurrences dont l'image est très similaire (même forme, donc aussi même fonte). Cela permettrait de corriger rapidement toute une catégorie d'erreurs [91].

Commandes d'édition Parfois, l'utilisateur désire retaper lui-même toute une portion de texte, peut-être parce que l'image est trop dégradée pour une reconnaissance automatique. Dans ce cas, nous avons proposé [91] que le système synchronise les caractères introduits avec l'image correspondante, de façon à positionner automatiquement le texte au bon endroit. Cette approche suppose qu'on connaisse la fonte et la découpe en ligne. Pour le texte reconnu par la machine, le texte devrait pouvoir être édité de manière conventionnelle. Afin de limiter les frappes au clavier, on pourrait offrir un mécanisme qui présente une liste d'alternatives pour le mot courant, à la manière du vérificateur orthographique `ispell` [207].

Commandes d'analyse Pour guider le système, l'utilisateur pourrait désigner une portion d'image à analyser et préciser les options de configuration. En fait, on peut même envisager un étiquetage logique préalable : le système serait alors capable d'ajuster l'outil d'OCR en conséquence. Comme nous l'avons déjà mentionné, il faudrait éviter que l'utilisateur corrige chaque occurrence d'une même erreur. Dans les cas de rejet, le logiciel peut constituer des groupes avec les caractères rejetés qui sont similaires, ce qui permet de corriger toutes les occurrence d'un seul coup. Une solution alternative, qui n'est pas limitée aux caractères rejetés, consiste à déclencher une recherche automatique de cas similaires, après chaque correction introduite par l'utilisateur (cf. section 8.6).

3.5.4 Etiquetage logique assisté

Nature de la tâche L'étiquetage sert à construire une structure logique (titre, auteur, date...) au-dessus des entités physiques (lignes, mots...). Le nom des étiquettes et les contraintes d'assemblage sont véhiculés par le modèle de document. Ce modèle est davantage qu'un paramètre de configuration : il est la manifestation de la structure générique que l'utilisateur élabore mentalement.

Présentation Nous conseillons de présenter la hiérarchie logique dans une vue séparée, à l'instar de l'arbre physique sur la figure 3.9. Il faudrait aussi donner accès au modèle, par exemple en présentant dans une

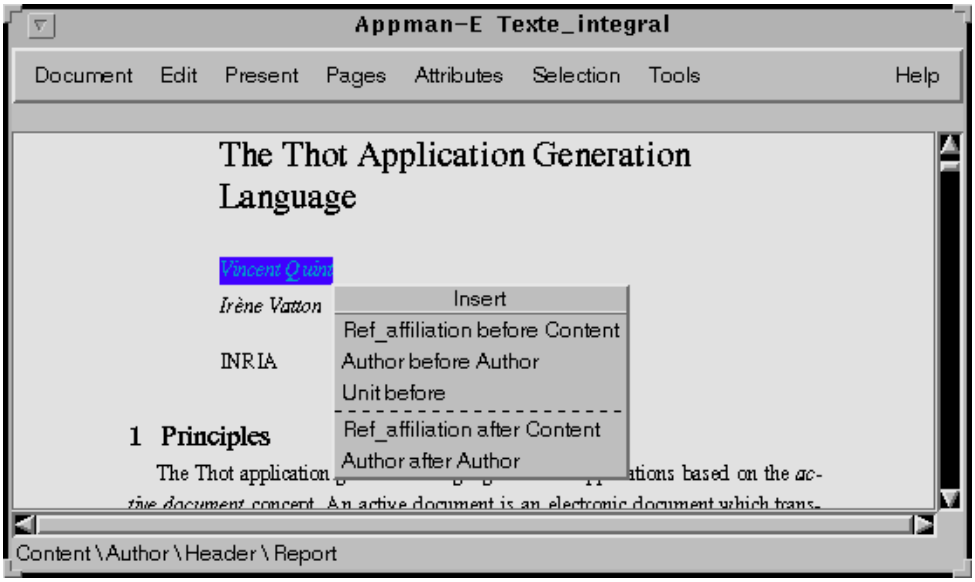


Figure 3.14 : Menu d'insertion contextuel et chemin vers la racine dans Thot.

fenêtre les diagrammes syntaxiques (les règles SGML ne sont lisibles que par un spécialiste). La vue affichant les images de l'échantillon peut aussi prévoir un mode (p. ex. une lentille magique) qui affiche le pourtour des entités logiques et le nom de l'étiquette associée. Une alternative serait d'afficher le chemin qui part de la racine du document jusqu'à la position courante de la souris. La figure 3.14 illustre cette fonctionnalité dans l'éditeur Thot [173] : le bas de la fenêtre nous indique que l'entité courante est du type «auteur», et qu'elle appartient à l'en-tête du rapport.

Commandes d'édition L'édition de la structure logique est analogue à la segmentation, moyennant deux différences : (i) la surface occupée par une entité logique peut être discontinue; (ii) en plus de la délimitation directe et du regroupement de fils, l'entité logique peut aussi être définie par un ensemble de composants physiques. En se basant sur le modèle et sur le contexte, le système est capable de restreindre les étiquettes possibles, à l'instar du menu d'insertion dans l'éditeur Thot[1] (cf. figure 3.14).

Commandes d'analyse Un analyseur de structure logique admet parfois un facteur de tolérance, selon que les solutions proposées doivent respecter strictement le modèle de document. Le système devrait en tout cas offrir une commande pour proposer une nouvelle interprétation en cas d'invalidation manuelle.

3.6 Incidences sur l'architecture logicielle

L'approche assistée modifie les hypothèses de travail lors de la réalisation d'une application d'analyse d'images de documents. Cette section résume les principales conséquences de l'approche assistée sur l'architecture logicielle des systèmes de reconnaissance.

3.6.1 Architecture centrée sur l'utilisateur

L'*interactivité* forte prônée par *CIDRE* impose des contraintes à l'architecture logicielle. Ainsi nous cherchons à supporter un fonctionnement asynchrone, qui autorise l'utilisateur à manifester ses intentions quel que soit l'état du programme, par exemple lorsqu'une analyse automatique est en cours. La modélisation informatique doit permettre aux événements de l'interface homme-machine d'être redirigés jusqu'au coeur du moteur d'analyse. Il ne suffit donc pas d'impliquer l'utilisateur dans la visualisation des résultats. C'est tout le schéma d'exécution qui est à remettre en question.

De plus, il ne faut pas sous-estimer l'effort de programmation supplémentaire autour de l'interface graphique. L'équipe de développement risque bien d'y consacrer la plus grande partie de son temps. Cette situation modifie les critères de décision sur le plan de la gestion de projet, par exemple en ce qui concerne le choix

[1] Notons que le calcul des étiquettes possibles dans un contexte donné sera différent, car contrairement à Thot, l'étiquetage logique devrait être aussi possible de manière ascendante.

de l'environnement de programmation.

3.6.2 Charpente réutilisable

L'*adaptabilité* soutenue par *CIDRE* influence aussi les aspects architecturaux. Notre principe général consiste à retarder la plupart des choix jusqu'en phase d'exploitation, afin de ne figer qu'un minimum de propriétés du système de reconnaissance. C'est naturellement le cas pour les paramètres de configuration des analyseurs automatiques, que l'on doit pouvoir modifier lors de l'exécution. Un raisonnement analogue s'applique par extension au *schéma d'exécution* lui-même. Par exemple, la phase d'OCR précède habituellement la reconnaissance de structure logique. Mais avec certains types de documents, des indications sur la segmentation et la fonte suffisent pour diriger l'étiquetage logique; l'OCR peut donc être lancé ultérieurement, et profiter du modèle de document, par exemple pour restreindre l'alphabet.

Dans un certain sens, les moyens pour favoriser l'adaptabilité servent en même temps un objectif de *réutilisabilité* logicielle. En effet, le concepteur est encouragé à construire un noyau de composants communs à plusieurs applications, puisque la spécialisation est reconnue comme une caractéristique importante du programme.

Un dernier point concerne le respect des temps de réponse dans un programme interactif. En effet, nous constatons que les techniques d'analyse d'images de documents sont souvent gourmandes en calcul. La lenteur d'un système de reconnaissance s'appréhende différemment selon que l'approche est automatique ou assistée :

- Si un programme entièrement *automatique* est trop lent, on risque un gaspillage de CPU, ce qui n'est plus vraiment grave de nos jours. Une parade simple et générale consiste à répartir les documents à traiter sur plusieurs ordinateurs.
- Si un programme *interactif* est trop lent, on risque un gaspillage de main-d'oeuvre, ce qui est économiquement plus pénalisant. Le parallélisme, que nous considérons comme une source essentielle d'accélération, ne peut être exploité que si l'architecture du programme prévoit les mécanismes nécessaires.

Nous estimons donc que l'adaptation du programme aux ressources informatiques disponibles mérite également d'être considérée. C'est pourquoi nous proposons une architecture capable d'exploiter un réseau de processeurs. Avec cette approche, les temps de réponse de l'environnement assisté devraient pouvoir être raccourcis en ajoutant les ressources de calcul nécessaires, sans adapter le code.

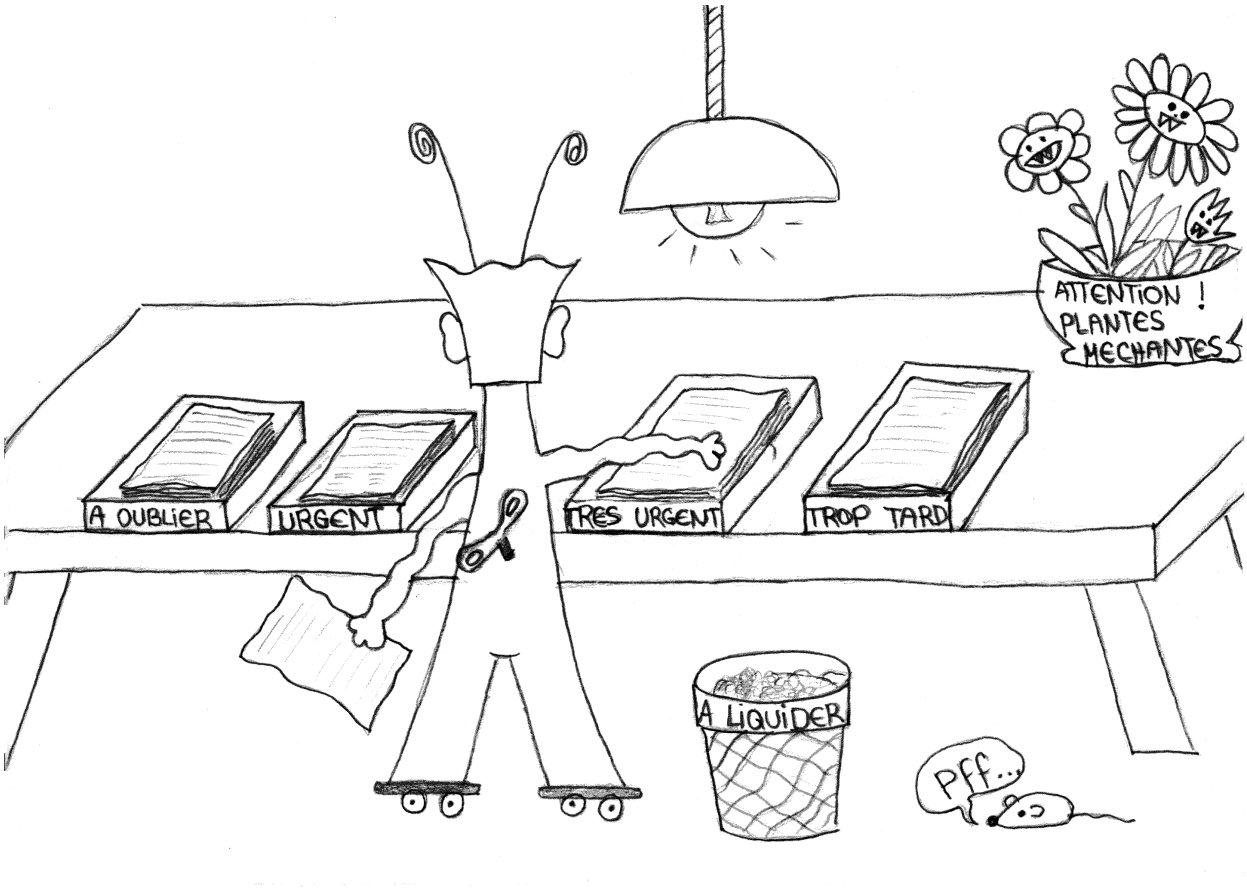
Conclusion

La revendication d'un environnement de reconnaissance assistée remet en question une hypothèse de base des systèmes de reconnaissance. Sur le plan de l'ergonomie, la quête d'une véritable coopération homme-machine en reconnaissance de documents est surtout une affaire de tentatives successives, testées dans des situations concrètes. Ce qui est certain, c'est que l'architecture logicielle va jouer un rôle important pour supporter l'interactivité et mettre les capacités de la machine au service des besoins de l'utilisateur. En fait, les conséquences de l'approche assistée sur l'architecture logicielle sont si profondes que ce thème est devenu l'objet principal de la présente thèse.

Le chapitre suivant aborde la conception d'une architecture homogène, en considérant la problématique sous l'angle de la gestion des données.

Chapitre 4

Gestion des données



Tout programme se caractérise selon deux axes : les données et le contrôle. Ce chapitre aborde la conception d'un système de reconnaissance de documents assisté en étudiant l'architecture des données. Nous commençons par établir un inventaire général des informations que le système doit manipuler. Cet ensemble de données est ensuite structuré, de façon à en formuler une représentation adéquate. Finalement, nous concrétisons nos propositions en définissant des structures de données, dérivées du standard DAFS [174].

Nos propositions constituent un cadre pour le développement d'applications. Nous ne prétendons pas que notre modélisation des données peut toujours être utilisée sans y apporter quelques retouches dictées par les besoins de l'application. De même, nous ne détaillons pas les représentations internes utilisées par les algorithmes d'analyse, mais nous nous concentrons sur la nature des informations manipulées en définissant une base commune pour toutes les étapes d'analyse.

4.1 Inventaire des informations à intégrer

Les informations à gérer dans un système de reconnaissance de documents assistée forment un ensemble complexe. Etablir un inventaire permet de mieux comprendre les liens qui les unissent, donc les principaux éléments d'une structuration adéquate.

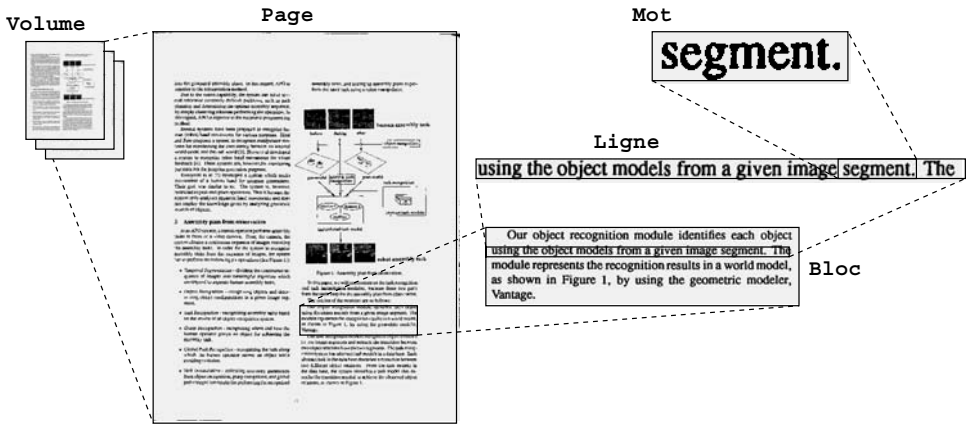


Figure 4.1 : Quelques types d'entités physiques.

4.1.1 Description de documents

L'ensemble de données le plus naturel concerne l'objet même du processus de reconnaissance, à savoir le document. C'est aussi dans les structures de documents que l'effort de formalisation a été le plus intense [90, 71, 99, 172], sans toutefois parvenir à un consensus. Une première distinction sépare classes et instances de documents. De manière orthogonale, les documents véhiculent deux formes d'information structurée, selon qu'on s'intéresse au contenu sémantique ou aux aspects de présentation. L'éditeur Thot [173] figure parmi les logiciels qui gèrent très proprement ces deux axes (générique/spécifique, et logique/physique).

Structure générique Le document générique décrit le modèle de toute une classe de documents similaires, c.-à-d. qui respectent les mêmes critères de construction. La classe des articles spécifiée pour la soumission à la conférence ICDAR'97 est un exemple de document générique. Un modèle de documents s'exprime en général à l'aide de règles de syntaxe et de présentation, comme dans Thot [173]. Toutefois, dans les systèmes de reconnaissance, ces informations sont souvent encodées sous un format qui facilite les opérations d'analyse syntaxique et de mise en correspondance [86, 44, 38].

Structure spécifique La structure spécifique correspond à un exemplaire de document. Les quatre pages du premier article publié dans les actes de la conférence ICDAR'97 constituent un exemple de structure spécifique. Le processus de reconnaissance s'attache à structurer un document spécifique, en partant d'une séquence d'images de page.

Structure logique La structure logique reflète l'interprétation du message perçu par le lecteur, à l'aide d'un étiquetage symbolique (auteur, résumé, phrase, mot-clé, etc.). Les étiquettes sont définies en fonction de l'application, en général par l'utilisateur lui-même. Comme Hu [86], nous distinguons plusieurs types d'étiquettes : le document, la partie, le fragment, la chaîne et le caractère. Les éléments logiques sont liés par des relations de composition, d'ordonnancement, ou de référence (p. ex. entre une citation et l'entrée correspondante dans la bibliographie). La structure logique générique décrit des règles de construction valables pour tous les documents de la classe. La structure logique spécifique décrit l'organisation conceptuelle d'un exemplaire.

Structure physique La structure physique décrit le formattage que subit le document; elle exprime l'organisation spatiale et la morphologie des éléments de présentation (colonnes, lignes, etc.). La structuration physique que nous adoptons est destinée à être reconstruite par analyse des images de pages, puis à servir de base à l'interprétation logique. Il peut donc y avoir des divergences avec les conventions adoptées par les outils d'édition (par exemple la notion de *boîte* en TEX [110]), puisque la démarche est inverse. Nos types de composants physiques reprennent essentiellement la terminologie d'Azokly [10]. La figure 4.1 illustre les entités intermédiaires depuis le volume multi-pages, jusqu'à un mot d'une ligne. Nous tenons aussi compte des entités non textuelles, comme les photographies, les graphiques, les tableaux, ou les formules mathématiques contenant des symboles. Outre le type des entités et leur position dans l'image, les informations physiques comprennent des liens d'ordonnancement et de composition, la description des fontes, et le mode d'alignement (p. ex. centré ou justifié). Une structure physique générique décrit les règles de formattage. Une structure physique spécifique décrit la mise en page d'un échantillon.

4.1.2 Etat de l'interface homme-machine

Les descriptions de documents se chargent de représenter l'ensemble des informations dans une solution finale, mais l'utilisateur a aussi besoin d'accéder à d'autres informations pour superviser la session de reconnaissance. Comme pour toute application interactive, la mise en oeuvre de l'interface graphique nécessite sa propre gestion des données. Dans le cas de la reconnaissance de documents, nous pensons notamment aux éléments suivants :

- options de visualisation, p. ex. pour afficher seulement le texte reconnu;
- mise en évidence des résultats selon leur statut (certifié, douteux, inconnu...);
- description des commandes associées à chaque élément graphique, via des menus contextuels;
- gestion de plusieurs modes de fonctionnement, p. ex. pour la segmentation manuelle;
- détermination d'une entité courante, comme le numéro de page;
- support pour défaire ou refaire les dernières opérations (undo/redo);
- notion de sélection, notamment pour supporter les opérations de copier/coller, ou pour y appliquer une commande.

En fait, il faut s'attendre à ce que la gestion du dialogue homme-machine implique un gros effort de programmation, et monopolise les ressources des développeurs. Les fonctionnalités manuelles et automatiques sont de fait intimement liées. Toutefois, le mélange des deux niveaux de programmation nuit à la lisibilité du code. Il faudrait donc s'efforcer d'organiser les sources du programme avec le souci de bien séparer l'interface homme-machine et le moteur d'analyse.

4.1.3 Configuration des analyseurs

De même que l'utilisateur est plongé dans un certain contexte de travail, ainsi en va-t-il pour les outils d'analyse automatique qui peuvent selon les cas être configurés avec de nombreux paramètres, par exemple :

- fichiers d'apprentissage pour un OCR;
- seuils pour la segmentation;
- modèles de polices pour la reconnaissance de fontes;
- options de pré-traitements d'images.

Cette paramétrisation est toujours déterminante pour atteindre une qualité suffisante dans les résultats trouvés. Afin de faciliter la reconfiguration du logiciel lors d'un changement des conditions d'utilisation, il est essentiel d'isoler les options de configuration, par rapport à la partie fixe de l'algorithmique (cf. section 7.1.2). C'est particulièrement important dans le projet *CIDRE*, puisque l'accent est mis sur l'adaptabilité du système, et sur l'enrichissement des connaissances par apprentissage.

Notons que la terminologie varie suivant la nature des informations qui servent à configurer les traitements. Ainsi on parle volontiers de *base de connaissances* lorsque les analyseurs accumulent de l'expérience, gardent une trace des exemples appris, ou consultent une base de données, comme c'est le cas avec le lexique utilisé par un OCR. Dans le cas où les possibilités de configuration se limitent à déterminer des options ou une poignée de valeurs, tels que les seuils de segmentation, on utilise plutôt le terme de *paramétrage*.

4.1.4 Progression dans l'espace d'analyse

L'état courant d'une session de reconnaissance est également caractérisé par un contexte d'exécution, qui décrit l'avancement du travail par rapport au flux d'instructions prévu par le programme complet.

Les renseignements sur la progression dans l'espace d'analyse peuvent être exploités à plusieurs usages. Dans l'interface homme-machine, ils servent à mettre en évidence les informations les plus importantes, ou qui méritent une validation de la part de l'utilisateur. Dans le moteur d'analyse, la connaissance des étapes précédentes permet d'appliquer les règles d'enchaînement des opérations.

Sur le plan des données à gérer, l'accomplissement des différentes étapes se manifeste bien sûr dans la solution courante, mais peut aussi être représenté explicitement sous forme de trace.

Le cheminement dans l'espace d'analyse peut accumuler des informations complémentaires à la solution courante :

- des résultats intermédiaires, comme la segmentation en composantes connexes;
- une liste d'hypothèses alternatives;
- la confiance accordée aux résultats trouvés;
- l'origine d'un résultat, selon que celui-ci est inconnu, confirmé par l'utilisateur, ou proposé par un analyseur automatique.

La connaissance des traitements déjà accomplis peut être dissimulée dans l'algorithmique. Par exemple, si un programme d'analyse prévoit d'exécuter séquentiellement la reconnaissance de fontes puis l'OCR, cette deuxième étape prend comme hypothèse que l'information de la fonte est disponible (mais pas certifiée). Dans notre démarche de briser l'unicité de la chaîne de traitement, nous recommandons de représenter *explicitement* les informations sur la progression de l'analyse, comme c'est par exemple le cas avec les architectures basées sur les tableaux noirs. De cette manière, on met en évidence la stratégie d'analyse, et on peut plus facilement la modifier. Il y a toutefois une limite à cette démarche, puisqu'à l'extrême, il faudrait conserver tout l'historique de la session en cours, afin d'adapter le plus finement possible la suite des opérations.

4.1.5 Entrées-sorties

Les sections précédentes ne donnent pas d'indications sur la permanence des informations de notre inventaire. Le choix des entités persistantes, c.-à-d. qui survivent après la terminaison du programme par stockage sur fichiers, donne des indications sur la manière d'organiser les structures de données. Le système se devra d'importer des images de pages, et d'exporter finalement la version structurée du document, mais ces fonctionnalités ne reflètent qu'un cahier des charges minimal.

Tout d'abord, la reconnaissance de documents assistée nous semble une tâche suffisamment complexe pour matérialiser le concept de *session* en tant qu'objet d'entrée-sortie. En effet, la reconnaissance d'un long document peut prendre plusieurs heures, et nous proposons d'offrir à l'utilisateur la possibilité de figer sur fichier l'état d'avancement de la reconnaissance, afin de reprendre le travail ultérieurement. L'idéal serait de sauvegarder un instantané complet du programme, pour restituer un contexte d'utilisation maximum. Dans une application où des traitements peuvent s'accomplir de manière concurrente, la détermination d'un état stable risque de buter sur des problèmes de synchronisation.

D'autre part, nous pensons que certaines parties des données gagneraient à être conservées séparément, pour des questions de réutilisabilité :

- Les *modèles de documents* font aussi partie de la solution, puisque c'est une des originalités du projet *CIDRE* de les reconstruire durant la reconnaissance des échantillons. On doit pouvoir charger un modèle de documents, si on désire reconnaître une instance d'une classe connue. Nous sommes partisans d'une granularité plus fine, car deux classes de documents distinctes peuvent manipuler en partie des structures communes, auquel cas il faudrait pouvoir récupérer un extrait d'un modèle dans un autre. Par exemple, des étiquettes logiques pour dénoter une date ou une entrée bibliographique, seraient communes à de nombreuses classes de documents. Cette idée de réutilisation de types d'entités s'applique aussi a fortiori à la mise en page : il y a de fortes chances de trouver des lignes de texte tout à fait similaires (quant à la fonte, la taille, et l'alignement) dans des documents variés.
- Même lorsqu'on change de classe de documents, les nouveaux échantillons peuvent présenter des caractéristiques similaires, comme le type de dégradation d'images (p. ex. les fax). Dans ce cas, on aimerait réutiliser la *configuration* de certains analyseurs, mais sans être obligé de conserver le même modèle de document.
- Du côté de l'interface graphique, on peut mettre en oeuvre un mécanisme de gestion de *préférences*, de façon à s'adapter aux habitudes de chaque utilisateur. Les préférences sont indépendantes de l'application de reconnaissance.

Ces remarques nous encouragent à organiser les informations de façon *modulaire*. L'objectif est de permettre de bénéficier ultérieurement de l'expérience acquise durant une session de reconnaissance, même lorsque le système est utilisé dans des contextes variés.

4.2 Organisation des données

Dans cette section, nous proposons une manière d'organiser l'ensemble des informations à manipuler dans le projet *CIDRE*. Tout d'abord, nous identifions les unités principales qui forment une ossature de base, sur laquelle se greffent les différents attributs. Ensuite, nous discutons les liens majeurs entre types d'entités.

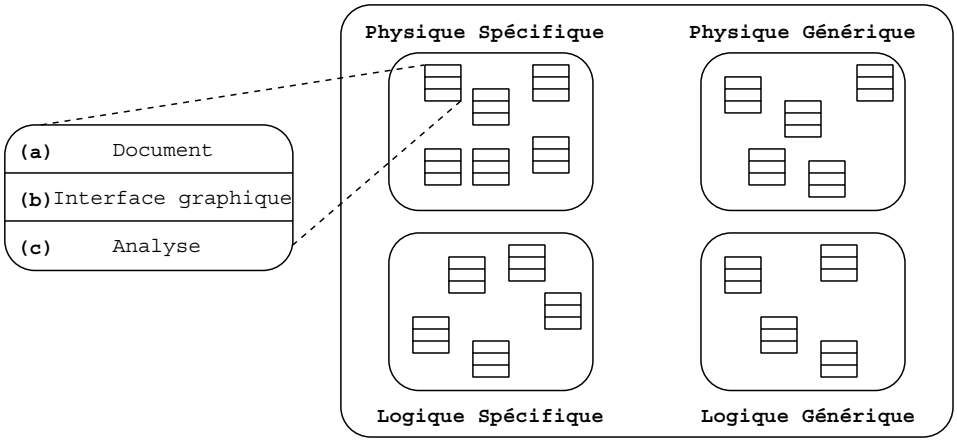


Figure 4.2 : Répartition des informations sur les entités dominantes.

4.2.1 Entités dominantes et attributs

Notre méthodologie consiste à reprendre l'inventaire des informations afin d'y identifier un ensemble de composants principaux, qui vont véhiculer les différentes données sous forme d'attributs.

La proposition que nous défendons ici est dérivée des structures de documents. L'idée revient à greffer toutes les informations du système sur les unités qui décrivent le document traité ou son modèle.

Comme le montre la figure 4.2, les *entités dominantes* de notre modélisation se rapportent uniquement à l'organisation du document. Nous distinguons quatre groupes, selon que les entités appartiennent à la structure physique ou logique, spécifique ou générique. Chaque catégorie définit plusieurs types d'entités, comme la *ligne* dans le cas physique ou la *partie* dans le cas logique. Avec cette approche, nous respectons les structures de documents, tout en les enrichissant avec les informations liées au système de reconnaissance.

Dans la structure que nous proposons, une entité ne décrit plus seulement son rôle dans le document, mais donne aussi des indications sur la manière de la présenter dans l'interface homme-machine, et sur la manière de conduire les analyses. Chaque entité véhicule donc, au moyen d'attributs, trois catégories d'informations :

(a) les données qui décrivent le rôle de l'entité en tant que portion du *document*;

(b) les informations liées à la gestion du dialogue homme-machine dans l'*interface graphique*;

(c) des indications pour le moteur d'*analyse*.

Suivant le type d'entité, les informations exprimées sous cette distinction se rapportent à différentes parties de l'inventaire des données. Par exemple, nous ne cherchons plus à définir globalement l'état de l'interface homme-machine ou la liste des commandes autorisées, mais plutôt à déterminer les compétences respectives de chaque entité quant à la gestion de l'interactivité. Avec notre optique, une entité physique spécifique de type «bloc» maintient les informations qui permettent au système de signaler sa présence par un rectangle dessiné sur la page, et d'intercepter les commandes d'édition que l'utilisateur désire lui faire subir. Une entité logique générique de type «titre» définit peut-être une vue qui présente la syntaxe que doivent respecter les instances. Si un OCR spécial doit s'appliquer sur les notes de bas de page, on attachera une telle indication sur l'entité physique générique représentant les «petites lignes».

On peut maintenant faire le lien avec les fonctionnalités évoquées sur la figure 3.3 de la section 3.3.1. La catégorie (a) de la figure 4.2 regroupera des attributs qui décrivent la *solution*. Quant aux catégories (b) et (c), elles concernent la *configuration* du système de reconnaissance, avec ses aspects interactifs et automatiques. Le tableau 4.1 montre en outre que nous appliquons la distinction entre spécifique et générique aussi bien au domaine du document qu'au niveau de la configuration :

- La *configuration spécifique* décrit l'état du système à un moment précis d'une exécution. Ces informations sont nécessaires pour piloter la session en cours, mais ne sont pas réutilisables dans un autre contexte. Du côté interactif, on pense à la sélection courante ou à la liste des dernières commandes. Du côté analytique, on trouve les indications sur la progression des analyses (historique, alternatives, taux de confiance).

	Solution	Configuration	
	Document	Interface graphique	Analyse
Spécifique	échantillon	état du dialogue	progression des analyses
Générique	modèle	préférences	bases de connaissances

Tableau 4.1 : Aspects spécifiques et génériques de la configuration du système.

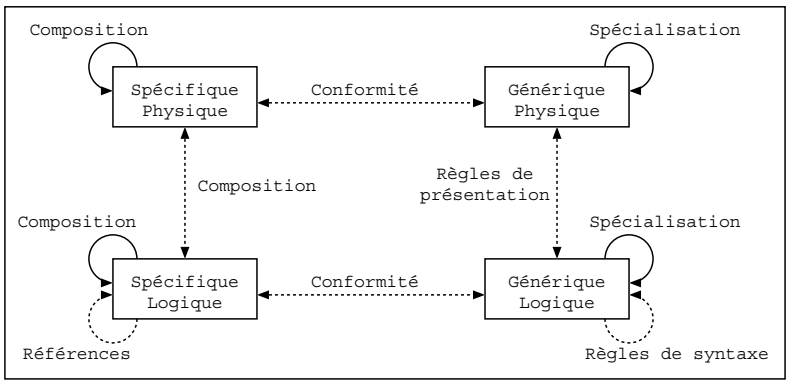


Figure 4.3 : Relations entre catégories d'entités.

- La *configuration générique* exprime des règles de paramétrisation du système. La portée de ces informations n'est pas limitée à la session en cours. Du côté interactif, il y a les préférences (options de visualisation). Du côté analytique, on range les bases de connaissances des analyseurs.

Nous n'allons pas imposer ici de format particulier pour la représentation des différentes connaissances, car le détail dépend de l'application. Le principe directeur que nous édictons consiste à adopter une *vue locale*, de manière à exhiber le rôle de chaque entité en regard des différentes catégories de données. Ainsi, les informations utiles à la gestion des analyses et du dialogue sont fragmentées, et réparties sur les constituants du document qui sont le plus concernés.

Globalement, notre approche, fondée sur une structuration solide du document, possède l'avantage d'établir un cadre homogène pour représenter les informations.

4.2.2 Relations dominantes

Quatre grands groupes d'entités dominantes ont été retenus. Nous proposons de diviser en six catégories les principales relations à établir entre ces groupes. Ces liens sont illustrés sur la figure 4.3.

Composition Les structures spécifiques définissent de manière naturelle une organisation hiérarchique entre entités. Par exemple, les lignes se décomposent en mots. Pour tisser des liens entre les domaines physique et logique qui ne sont pas en correspondance biunivoque, nous définissons en plus une *décomposition duale* : toute entité logique est associée à la plus petite entité physique qui la contient entièrement, et vice-versa. Sur la figure 4.4 par exemple, le titre de la publication est une entité logique, qui admet comme père dual une ligne de texte physique, et possède comme fils duals deux mots physiques. Cette notion de lien dual n'est pas rapportée dans la littérature, peut-être parce qu'elle ne n'apporte rien au génie documentaire. Par contre, à notre avis, le mécanisme de lien dual pourrait être utile durant la reconnaissance. Nous pensons surtout aux étapes intermédiaires où la structuration, tant physique que logique, est encore incomplète. En génie documentaire (p. ex. dans la norme ODA [7]), on définit les liens entre structure physique et logique à travers le partage des mêmes feuilles, à savoir les signes. Mais ces relations n'ont de sens que pour un document *entièrement structuré*. Au contraire, la décomposition duale permet toujours d'associer intimement les deux structures.

Références A l'intérieur de la structure logique, la hiérarchie naturelle peut être court-circuitée par des liens sémantiques particuliers, tels qu'un renvoi à une note de bas de page ou à une entrée bibliographique. Nous parlons alors de *liens de référence*.

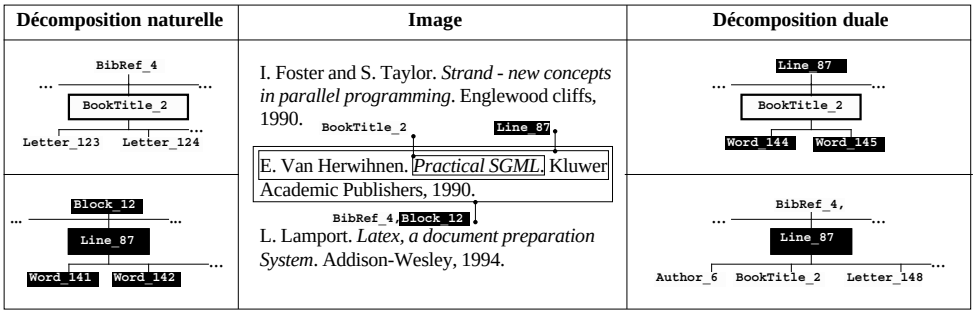


Figure 4.4 : Décompositions naturelle et duale des entités.

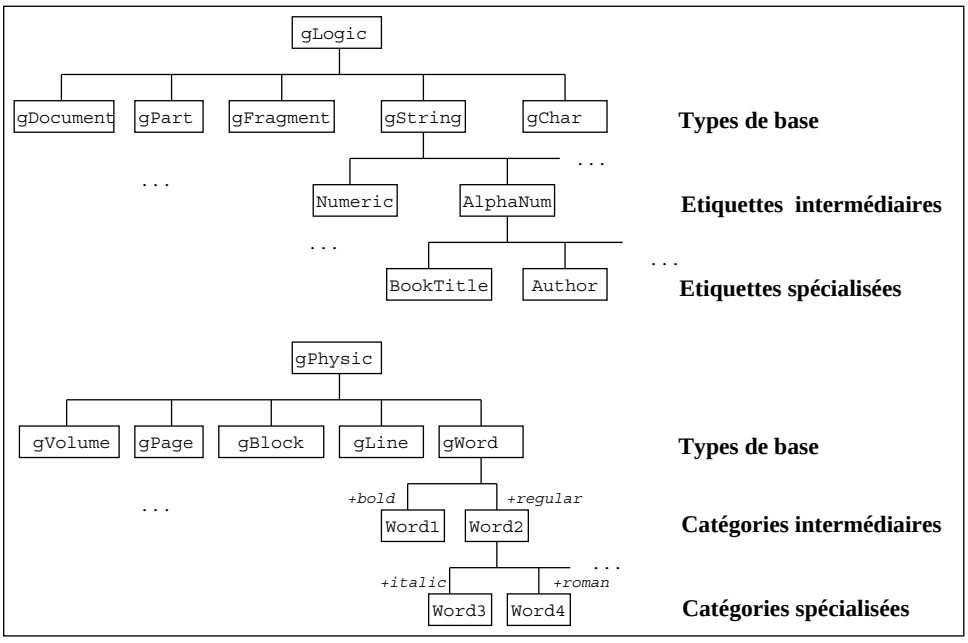


Figure 4.5 : Spécialisation d'étiquettes logiques et de catégories physiques.

Conformité Chaque entité spécifique est considérée comme une instance d'une entité générique. Dans notre proposition, la *conformité* relie une entité générique à chacune de ses instances dans l'exemplaire en cours. La relation entre classe d'entités et échantillons est de cardinalité 1:n.

Spécialisation Les entités génériques s'organisent par spécialisation successive des types de base. Les classes logiques sont désignées par une étiquette sémantique choisie par l'utilisateur. Les catégories physiques sont formées automatiquement par classement des entités spécifiques, pour constituer des groupes d'entités partageant des caractéristiques communes (p. ex. géométriques ou typographiques). Ces deux organisations, qui autorisent une gestion modulaire du modèle, sont illustrées sur la figure 4.5. A notre connaissance, cette notion de *catégorie intermédiaire* n'est pas utilisée dans les méthodes courantes de modélisation des structures génériques, mais nous pensons que c'est une extension intuitive qui pourrait augmenter la réutilisabilité des modèles de documents. Une modélisation rigoureuse devrait faire l'objet d'une nouvelle étude.

Règles de syntaxe Un modèle de documents définit des règles de construction qui précisent la syntaxe des éléments logiques autorisés. Ces contraintes syntaxiques, qu'elles prennent la forme d'une grammaire [90, 173] (p. ex. une DTD pour SGML) ou d'informations statistiques [38], induisent des relations de construction complexes entre entités génériques logiques. Un modèle de documents va par exemple exprimer qu'une entrée bibliographique commence par la liste d'auteurs, suivie du titre et de l'année, et peut ensuite contenir divers éléments optionnels comme le nom de la revue ou l'indication des pages. Nous retenons dans l'architecture des données le concept de règle syntaxique, mais sans imposer un formalisme particulier.

Règles de présentation Un modèle de document définit également des règles de présentation. Ces directives sur le formattage s'encodent parfois via des grammaires attribuées [90]. Dans notre modélisation

des données, ces règles de présentation expriment une réécriture des entités génériques logiques en termes d'entités génériques physiques. De ce point de vue, notre approche est compatible avec le modèle de Rolf Brugger [38].

4.3 Structures de données

Les sections précédentes ont développé une proposition abstraite pour organiser les informations dans le projet *CIDRE*. Nous allons franchir un pas supplémentaire en spécifiant la représentation interne. Concrètement, nous construisons nos structures de données à l'aide du standard DAFS [174, 62].

4.3.1 Représentation uniforme

On serait tenté de considérer la représentation des données comme un détail d'implémentation. Mais dans le contexte du projet *CIDRE*, plusieurs arguments militent en faveur d'une modélisation préalable des structures de données.

Les différents outils d'analyse automatique (OCR, segmentation, reconnaissance de fonte etc.) sont interdépendants. En effet, un résultat partiel n'est en général pas du ressort d'une tâche unique. Ainsi les coordonnées d'un caractère peuvent provenir d'un regroupement/éclatement de composantes connexes, mais elles sont aussi attestées par l'OCR (qui procède à une autre segmentation). De même, on peut disposer d'une reconnaissance de fonte, mais le taux d'erreur d'un OCR monofonte est aussi a posteriori une indication sur la confiance à accorder à l'interprétation typographique. On est ainsi amené à préciser le rôle de chaque donnée (et de chaque requête d'analyse) si l'on tient à ce que les résultats soient mis en commun de manière sensée [45], c.-à-d. en remédiant aux éventuelles incohérences. C'est le premier pas vers différentes formes de coopération et de compétition entre sources de connaissances (cf. section 7.2).

Non seulement les outils d'analyse participent conjointement à la construction d'une solution, mais surtout l'incertitude qui enveloppe chaque résultat partiel obligera de mettre en jeu des stratégies sophistiquées pour explorer l'espace des solutions. Ainsi la solution courante évolue bien sûr par l'incorporation de nouveaux résultats, mais aussi par la remise en cause de composants existants (on touche là le coeur même du moteur d'évolution d'une reconnaissance de documents). Ces révisions locales successives ne seront performantes que si les données se trouvent sous une forme adaptée et homogène.

La revalorisation du rôle de l'utilisateur passe par une présentation soignée qui prévoit des primitives de navigation, mais aussi de modification de la solution en cours. En fait, le pilotage d'une session de reconnaissance nécessite un accès adéquat à tous ses constituants. Ainsi les résultats devront être interfacés avec les commandes d'édition manuelle, ce qui requiert une implémentation soignée.

Enfin, il s'agit de préparer le terrain pour différentes extensions, telles que l'utilisation de nouveaux outils d'analyse (p. ex. un vérificateur lexical), ou la traduction depuis/vers d'autres formats de documents. Ces extensions seront facilitées avec une représentation uniforme de la session en cours.

4.3.2 Paquetage DAFS

DAFS [174, 62], développé par RAF Technology, se veut un support pour le ré-encodage d'images de documents, en définissant un type abstrait sous forme de librairie C et la spécification d'un format de fichier. A notre avis DAFS devient un standard [1] en la matière conformément à l'un de ses objectifs. Notons tout de même qu'il existe d'autres formats comparables : par exemple, le système PRASAD [123] (cf. section 2.1) utilise ODIL [124], un langage de balisage basé sur SGML [80], qui sert à représenter la mise en pages des documents durant la reconnaissance.

A la base, DAFS est tout d'abord un gestionnaire d'arbre, étant donné que c'est la structure hiérarchique qui est la mieux adaptée à la gestion des documents structurés. Une *entité DAFS* est en fait un noeud d'arbre, qui représente typiquement une unité du document à reconnaître, comme une ligne de texte ou un caractère. Les auteurs du format ont étendu ce concept en distinguant les noeuds ET et les noeuds OU, dont les fils ne symbolisent pas la composition, mais des points de vues alternatifs sur l'entité. Cette distinction est accompagnée d'un mécanisme d'*emprunt*, qui permet à deux noeuds différents de partager un fils commun, ce qui dépasse la hiérarchie stricte. Dans la figure 4.6 par exemple, la structure DAFS véhicule deux hypothèses sur le contenu d'un mot; toutefois, les lettres communes aux deux interprétations ne sont pas dupliquées,

[1] depuis la rédaction de la thèse, il semble que l'avenir de DAFS soit plus incertain...

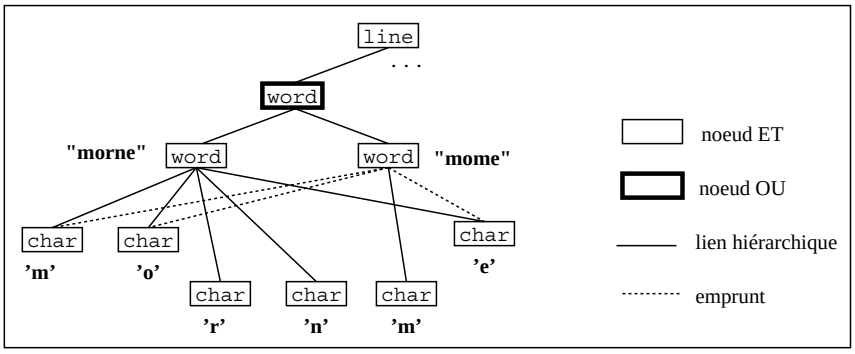


Figure 4.6 : Arbre ET/OU et mécanisme d'emprunt en DAFS.

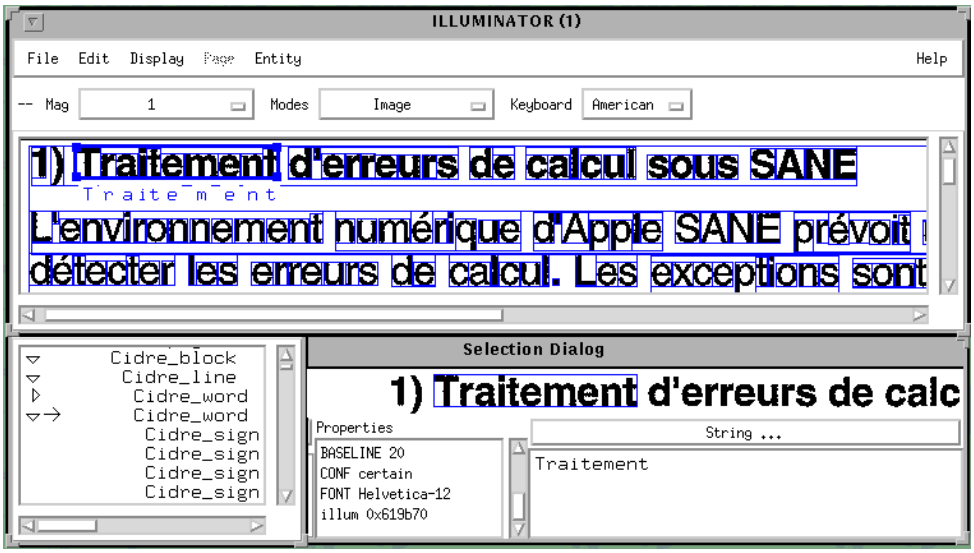


Figure 4.7 : Visualisation et édition avec Illuminator.

mais partagées par les deux alternatives, ce qui évite la redondance. Grâce aux noeuds OU et à l'emprunt, DAFS permet entre autres de cumuler dans la même structure la découpe physique et la découpe logique, ou de gérer plusieurs ordres de lecture.

Cette gestion d'arbre est complétée par une gestion d'images et de texte. Chaque noeud peut faire référence à une *image* (par défaut celle du père) et y occuper une surface déterminée par un polygone. De plus, chaque entité peut posséder un *contenu*, avec des solutions spécialement prévues pour traiter les contenus textuels (ASCII ou même UNICODE).

Finalement, le format est extensible puisqu'on peut définir des *propriétés*, identifiées par des chaînes de caractères, qui permettent d'associer toutes sortes de données aux entités, p. ex. la fonte. De plus, on a l'occasion de définir de nouveaux *types d'entités*. A titre d'exemple, certaines entités classiques (blocs, lignes etc.) sont prédéfinies.

Le paquetage complet, disponible sur le réseau depuis le serveur DIMUND [59], contient aussi une application appelée Illuminator, qui est un éditeur interactif de documents DAFS. La figure 4.7 illustre quelques fonctionnalités d'Illuminator, comme la superposition du texte sur l'image, la visualisation des propriétés, ou la vue hiérarchique. D'autres facilités sont offertes, par exemple pour visualiser rapidement les zones douteuses, ou toutes les occurrences d'un caractère donné (cf. section 3.5).

Globalement, le format DAFS possède tous les atouts d'un support adéquat pour la reconnaissance de documents. C'est pourquoi nous préconisons l'usage de DAFS pour gérer les données dans le projet *CIDRE*.

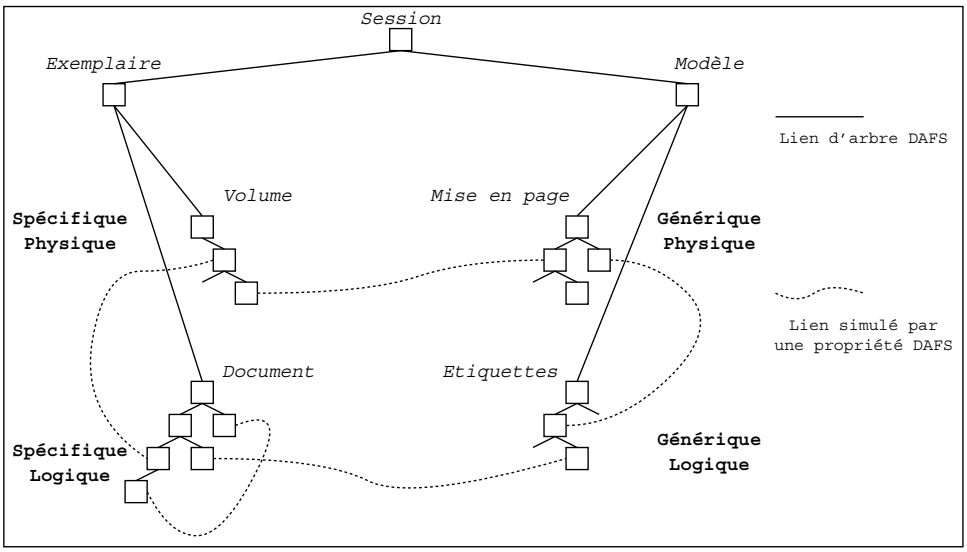


Figure 4.8 : Relations entre entités et représentation DAFS.

4.3.3 Spécification basée sur DAFS

Traduire en DAFS un ensemble de données comme celui de la section 4.1 revient à répondre aux questions suivantes :

- Quelles unités d'information représenter par des noeuds d'arbre?
- Quelle sémantique donner aux relations hiérarchiques entre noeuds?
- Comment convertir le reste des informations sous forme de propriétés DAFS?

L'organisation abstraite discutée dans la section 4.2 préfigure certains choix. Nous avons ainsi décidé de construire l'arborescence DAFS autour des entités dominantes.

Comme illustré sur la figure 4.8, nous avons choisi respectivement les relations de composition et les relations de spécialisation pour supporter les liens hiérarchiques de la structure DAFS. Les autres relations, qui étaient en traitillé sur la figure 4.3, doivent être simulées car elles n'appartiennent pas à l'arbre de base. Dans ce but, il faut un mécanisme qui autorise la désignation d'un noeud DAFS hors de son voisinage direct. Nous proposons une solution simple qui consiste à accorder un identificateur unique (p. ex. un entier) à chaque sommet de l'arbre. La librairie DAFS permet de retrouver un noeud étiqueté par une certaine propriété. Les relations de décomposition duale, de conformité et de référence sont directement représentées avec des propriétés qui contiennent soit un identificateur de noeud, soit une liste d'identificateurs. Quel que soit le format des règles de syntaxe et de présentation, on peut aussi utiliser ces identificateurs pour désigner les entités qui y participent.

Quant à la conversion en DAFS des différentes catégories d'attributs, nous n'avons pas trouvé de solution idéale. Il faudra choisir le mécanisme de traduction de cas en cas. A défaut d'une représentation élégante, on pourra toujours coder les informations sous forme de chaînes de caractères.

Dans le prototype (cf. section 6.4) que nous avons développé, seule une partie de cette spécification a été mise en oeuvre. De toutes façons, certains aspects des données à gérer vont dépendre de l'application visée, et des analyseurs utilisés. Pour notre part, nous avons surtout soigné la représentation du document spécifique, par exemple grâce à une convention dans le nommage des fontes (cf. section 7.3). Pour les informations du moteur d'analyse, nous avons essayé d'exploiter DAFS au maximum. Par exemple, nos outils d'OCR monofonte (cf. section 8.2) ou de post-correction d'OCR (cf. section 8.6) représentent leurs connaissances dans un arbre DAFS, et nous avons écrit des convertisseurs pour l'OCR ScanWorX [29], ou l'outil de segmentation d'Azokly [10]. Quant aux informations de l'interface graphique, c'est le choix de l'environnement de programmation qui a facilité la conversion. Comme notre interface graphique est pilotée par le langage de script Tcl-Tk [165], les informations correspondantes sont toujours codées par du texte, et peuvent sans problème être encapsulées dans des propriétés DAFS.

Au sujet de la représentation des règles de présentation et de syntaxe, nous sommes restés volontairement abstraits, en évitant de figer le modèle sous-jacent. Toutefois, notre architecture établit une base adéquate pour la modélisation proposée par Rolf Brugger [38], où les connaissances sont aussi associées aux entités

locales.

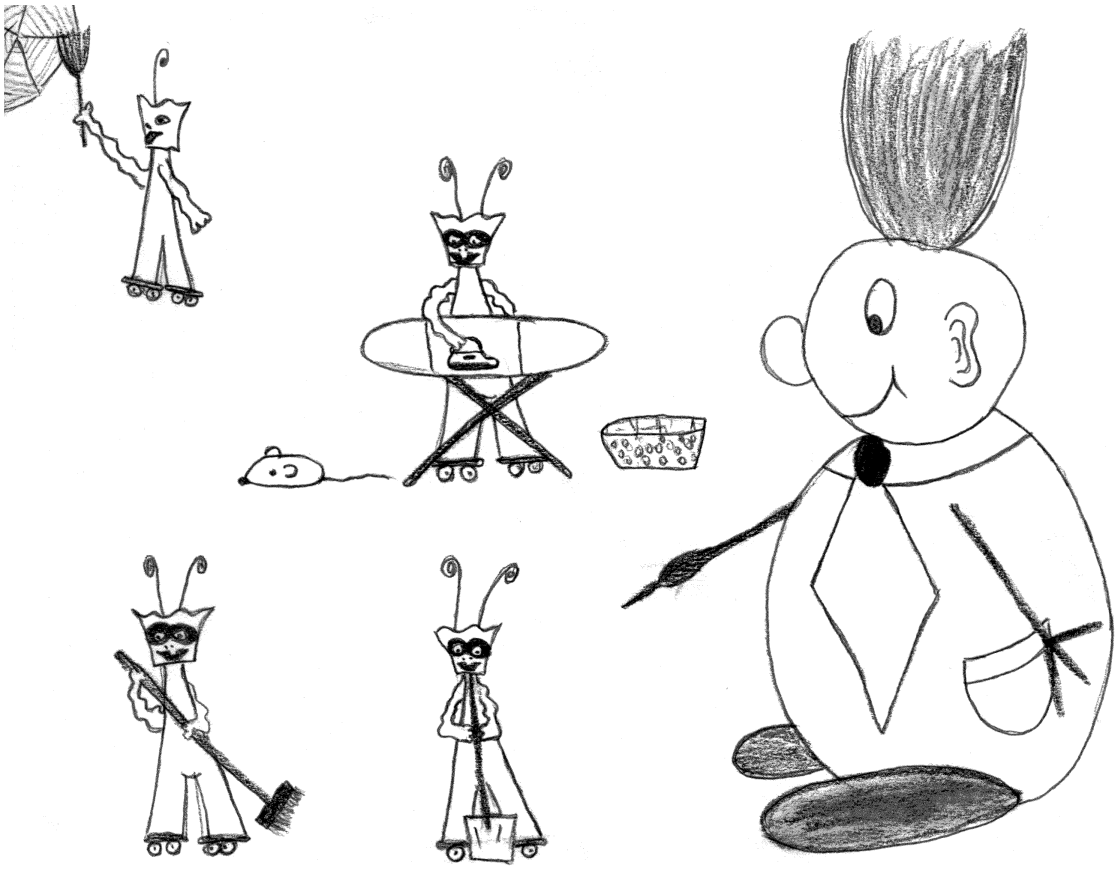
Conclusion

Un système de reconnaissance de documents assisté doit gérer une grande masse d'informations. Nous avons proposé une organisation des données qui étend la notion de document pour y greffer les indications nécessaires au processus de reconnaissance. Notre architecture des données s'articule autour des catégories d'entités définies en génie documentaire (physiques/logiques, spécifiques/génériques), et des relations naturelles qui les unissent. C'est sur cette structure que nous répartissons les informations propres au fonctionnement du système de reconnaissance. Cette approche autorise une gestion modulaire de tout ce que comprend une session de reconnaissance, comme des extraits d'un échantillon, d'un modèle de documents, de la stratégie d'analyse, ou de la configuration des outils. Le format de données DAFS, qui a été conçu pour l'analyse d'images de documents, nous a permis de mettre en oeuvre une partie de nos propositions. Le recours à DAFS apporte un élément de standardisation bienvenu pour la manipulation des structures de données.

Après avoir organisé les données, nous allons modéliser le contrôle dans notre architecture logicielle, en nous inspirant des systèmes multi-agents.

Chapitre 5

Conception en système multi-agents



Notre but est de concevoir une architecture logicielle adéquate pour la reconnaissance de documents assistée. Après l'organisation des données, nous allons maintenant aborder l'organisation du contrôle, et dessiner les grandes lignes de notre architecture. Il s'agit d'identifier les principaux composants d'un système de reconnaissance, et de spécifier leurs rôles respectifs. Par rapport aux architectures classiques, notre défi consiste d'une part à intégrer l'utilisateur comme un intervenant dans le processus de reconnaissance, et d'autre part à figer un minimum de choix quant à la stratégie d'analyse.

Pour conduire la découpe du système de reconnaissance, nous avons décidé de nous inspirer de l'intelligence artificielle distribuée. A notre avis, le déroulement d'une session de reconnaissance assistée met en jeu des phénomènes caractéristiques des systèmes multi-agents : imprévisibilité des événements, conflits, coopération, apprentissage, etc.

Nous considérons la modélisation en système multi-agents comme une étape conceptuelle, indépendante des choix de l'implémentation. C'est pourquoi nous évitons pour le moment d'entrer dans le détail des mécanismes qui vont concrétiser le comportement des entités de contrôle.

La première section développe les avantages apportés par l'intelligence artificielle distribuée, en particulier dans le contexte du projet *CIDRE*. Les sections 5.2 et 5.3 présentent les deux axes qui servent à partitionner les unités de contrôle. Cela aboutit à une description statique de chaque catégorie d'agent au sein de l'en-

semble de la population. Dans la section 5.4, nous donnons des indications sur l'évolution du système durant l'exécution.

5.1 Apport de l'intelligence artificielle distribuée

Les études en intelligence artificielle distribuée (IAD) [134, 212] portent sur les phénomènes collectifs engendrés par amalgame de comportements individuels. On parle de *systèmes d'agents autonomes*, au sens où les individus décident eux-mêmes de leur réaction face aux événements extérieurs, et des moyens d'adaptation à leur environnement.

L'IAD est une discipline récente de l'informatique, et souffre de défauts de jeunesse. Par exemple, il n'y a pas encore vraiment de consensus sur une définition stricte de l'IAD, des systèmes d'agents autonomes, ou de la résolution de problèmes distribuée. Vues de l'extérieur, certaines contributions semblent parfois tenir moins de la percée scientifique que de la guerre de religions. Au lieu de nous impliquer dans ces débats de fond, notre ambition se limite à saisir à la fois l'esprit général et quelques techniques concrètes, avec l'espoir qu'une définition d'agent intuitive plutôt que formelle n'en constitue pas un trop gros obstacle.

A notre avis, les paradigmes d'IAD s'appliquent surtout en phase de conception, alors qu'une réalisation s'exprime avec un vocabulaire plus concret (processus lourd, thread, Remote Procedure Call, interruption, etc.). C'est pourquoi nous utilisons le terme *agent* seulement en tant qu'*abstraction* pour désigner une entité soumise à son propre contrôle[1]. Dans un système multi-agents, le contrôle est décentralisé; les agents interagissent entre eux, mais ils tendent à se comporter de manière autonome. L'annexe A propose une définition plus précise d'un agent, en mettant l'accent sur l'indépendance du concept par rapport aux formes d'expression dans les langages de programmation.

Si nous nous inspirons de l'IAD pour concevoir une architecture *CIDRE*, c'est que certains de ses fondements nous semblent tout à fait indiqués dans le contexte de la reconnaissance de documents assistée. En effet, notre domaine d'application constitue un terrain propice aux paradigmes d'IAD. La suite de cette section évoque quelques caractéristiques de l'approche multi-agents, et motive leur utilité potentielle dans la problématique *CIDRE*.

Il ne serait pas très équitable de mettre en avant les avantages de l'approche IAD sans évoquer certains de ses inconvénients :

- le souci d'autonomie peut introduire de la redondance sur certaines informations;
- l'effort d'abstraction n'est pas toujours justifié : si on veut modéliser un simple compteur, parler d'*état mental*, de *stratégie* ou de *représentation du monde* semble un peu futile;
- la décentralisation du contrôle oblige à encoder la synchronisation de manière explicite.

5.1.1 Principe de localité

Par sa décentralisation du contrôle, l'approche IAD force le concepteur à *décomposer son application*, en identifiant des «centres de responsabilité». Cet aspect est a priori intéressant pour le projet *CIDRE* dont l'ampleur fait redouter une conception monolithique.

Une fois la découpe en agents établie, le souci d'autonomie incite à expliciter les interactions ou les dépendances avec le monde extérieur. Les *problèmes de coordination* et de coopération [164] sont alors affrontés directement, car la réflexion au niveau local remet en question chaque hypothèse sur le contexte d'exécution. Or, coordonner le recours aux différentes sources de connaissances constitue une difficulté majeure pour le projet *CIDRE*.

Répartir le contrôle peut aussi servir à séparer plusieurs *niveaux de programmation*. Dans notre cas, la mise en oeuvre d'un environnement de reconnaissance complet implique des classes de problèmes hétérogènes. On ne se trouve pas dans le même contexte de travail selon qu'on développe un algorithme de traitement d'images, une heuristique de satisfaction de contraintes, ou le pilotage de widgets. Une découpe multi-agents peut aider à cantonner chaque niveau de programmation de manière expressive.

La définition d'entités autonomes permet de mieux faire face à l'*imprévu*, puisqu'on cherche à réduire les hypothèses sur l'environnement des agents. Or, l'une des difficultés majeures en analyse de documents tient

[1] En IAD, l'autonomie est une propriété essentielle. Malheureusement, dans le langage courant, le terme «agent» évoque aussi un individu sous contrôle, qui reçoit des ordres.

au traitement des cas exceptionnels, qui prend sa source autant dans l'incertitude entourant chaque résultat que dans les anomalies des documents eux-mêmes. Dans un système multi-agents, la remise en cause locale d'un résultat ne doit pas perturber la progression du système dans sa globalité.

Enfin, le principe de localité encourage à exprimer des *traitements dépendants du contexte*, un phénomène courant en analyse de documents. Par exemple, la reconnaissance du texte d'un mot peut dépendre des traitements déjà subis par cette entité, ou des propriétés des mots voisins. Dans une approche monolithique, une trop grande dépendance au contexte est plutôt néfaste, car elle complique le schéma d'exécution en multipliant les embranchements possibles de l'algorithme. Mais avec une approche multi-agents, la situation est tout à fait naturelle, et justifie même la décentralisation du contrôle au niveau le plus approprié. En effet, le développeur qui adopte une vue locale est plus attentif à l'influence du contexte sur le comportement d'un individu, donc finalement du programme.

5.1.2 Non déterminisme

Nous parlons de comportement *non déterministe* [57] lorsque l'évolution d'un programme possède un caractère imprévisible. Certaines approches informatiques, notamment certains langages de programmation, autorisent l'expression d'une forme de non déterminisme. C'est le cas de l'approche multi-agents, puisqu'on n'impose pas un ordonnancement absolu, ni sur l'activation des agents, ni sur le traitement des interactions. Suivant l'implémentation d'un système multi-agents, le côté imprévisible peut être effectivement observable, par exemple en raison des aléas liés aux échanges de messages à travers le réseau.

Nous distinguons trois niveaux de motivation quant à l'usage du non déterminisme en programmation :

(a) éviter d'imposer une séquentialité arbitraire quand l'exécution des instructions est effectivement indépendante;

(b) éviter de coder un choix lorsqu'il n'y a pas de critère pour justifier la décision;

(c) étudier des formes de comportement inaccessibles à l'algorithmique purement déterministe [204].

Dans la situation (a), le recours au non déterminisme ne modifie en rien l'effet du programme, par exemple sur les résultats produits. Il s'agit juste d'épurer l'expression de l'algorithme de toute synchronisation inutile. Un compilateur peut exploiter cette propriété, car les parties de code marquées comme indépendantes peuvent ensuite être parallélisées.

Dans la situation (c) au contraire, on cherche à observer les effets du non déterminisme, en espérant même faire apparaître des propriétés qui ne pourraient pas être décrites de manière déterministe [54]. Nous pensons tout spécialement aux travaux sur la *vie artificielle*.

Pour justifier la situation intermédiaire (b), il faut rappeler que les problèmes informatiques se divisent en gros en deux catégories. D'un côté, il y a ceux dont on sait exprimer une solution algorithmique correcte et efficace. De l'autre côté, il y a les problèmes dont la résolution se base sur des connaissances partielles : propriétés de la solution, heuristiques, règles d'inférence. Plusieurs paradigmes de programmation peuvent servir à affronter cette deuxième catégorie, comme par exemple les systèmes experts, le modèle du tableau noir, ou la programmation par contraintes [191]. Dans ce contexte, les systèmes multi-agents constituent une alternative supplémentaire. Dans une approche heuristique d'un problème, il peut arriver qu'à une certaine étape de la résolution, on se retrouve en face de plusieurs alternatives, sans *aucun* moyen de déterminer la plus judicieuse. Dans ce cas, le non déterminisme permet de rendre compte de l'incapacité à choisir entre plusieurs comportements.

Or, nous prétendons qu'aucun algorithme déterministe ne résoudra la reconnaissance de documents dans sa généralité. En effet, l'expertise nécessaire ne peut que partiellement s'exprimer par un ensemble de règles.

Par rapport au projet *CIDRE*, nous voyons deux raisons d'abandonner un déterminisme absolu. D'une part, les *interventions de l'utilisateur* forment une importante source de non déterminisme. Le concept de reconnaissance assistée laisse une grande liberté sur les modalités du dialogue homme-machine. L'approche multi-agents permet justement de traiter les interactions homme-machine sans leur imposer un contexte d'exécution précis.

D'autre part, nous voulons offrir au concepteur la possibilité d'exprimer, le cas échéant, les situations du type (b). Evidemment, il ne suffit pas de prévoir le non déterminisme pour améliorer les résultats. Toutefois, l'approche non déterministe n'est a priori pas plus mauvaise que celles qui forcent le programmeur à figer des choix totalement arbitraires. Peut-être même qu'un aspect imprévisible rendrait le système de reconnaissance un peu plus surprenant, et plus agréable à utiliser.

A long terme, on peut toujours espérer que des situations du type (c) soient un jour exhibées en reconnaissance

de documents, et que les systèmes du futur tireront profit de comportements émergents.

5.1.3 Aspects de génie logiciel

Sous certains aspects, l'IAD rejoint les objectifs de génie logiciel. A notre connaissance, les connexions entre ces deux disciplines n'ont pas fait l'objet d'une étude approfondie.

En basant la modélisation sur une collection d'entités semi-autonomes, l'approche multi-agents favorise la *modularité*. L'application est décomposée en entités plus simples à maintenir. Le principe de localité se reflétera aussi dans les sources du programme. En fait, l'approche multi-agents se justifie également sur le long terme : à mesure que le système croît, il est plus aisé de maintenir l'expressivité du système lorsqu'on réfléchit au niveau local.

La *réutilisabilité* est aussi encouragée. En cherchant à rendre les agents autonomes, on accroît leur capacité à être utilisés dans différents contextes. De plus, comme la modélisation ne fait que peu de suppositions sur la nature des agents, on ouvre la porte à l'intégration d'agents hétérogènes. On peut donc espérer faire cohabiter des entités écrites dans différents langages de programmation. C'est d'ailleurs un avantage exploité par notre plateforme (cf. section 6.3).

La décentralisation du contrôle au niveau conceptuel permet ensuite une *parallélisation effective*, où l'exécution des agents est répartie sur plusieurs processeurs. Suivant les cas, cette propriété constitue un facteur d'accélération appréciable. C'est particulièrement intéressant en reconnaissance de documents, où les temps de réponse sont difficiles à respecter en raison de la complexité des techniques d'analyse (cf. section 3.6.2). Notre plateforme d'implémentation permet d'exploiter le parallélisme au niveau des tâches et des données. (cf. sections 6.1.2 et 6.2.3).

L'approche décentralisée facilite aussi la transition vers des *applications distribuées*. On peut ainsi concevoir, sur le système de base, un environnement multi-utilisateurs, ou une architecture clients-serveur. C'est un argument à ne pas négliger en regard des applications orientées vers le Web.

5.2 Dichotomie figuratif/opératoire

Notre découpe multi-agents est déterminée par une distinction entre deux types de connaissances :

- Les *connaissances figuratives* concernent les propriétés et les relations décrivant les éléments d'un problème.
- Les *connaissances opératoires* se réfèrent aux outils de traitements et d'analyse des ces éléments.

En d'autres termes, les connaissances figuratives décrivent les données du problème en termes d'objets et d'attributs, et les connaissances opératoires décrivent des règles de transformation, sous forme de méthodes d'analyse.

Sur la base de cette distinction, nous définissons grossièrement deux catégories d'agents, illustrées sur la figure 5.1. D'une part, les *agents-données* modélisent les connaissances figuratives; ce sont des gestionnaires de résultats, qui se chargent de faire évoluer la solution courante. D'autre part, les *agents spécialistes* modélisent les connaissances opératoires; ils représentent des domaines d'expertise, et sont sollicités pour analyser une partie de la solution. La génération de nouveaux agents-données, symbolisée par des traitillés sur la figure 5.1, est une opération courante durant l'évolution du système. Les relations entre les deux sous-ensembles d'agents sont essentiellement de nature client-serveur. Les agents-données soumettent des requêtes aux spécialistes, qui traitent une requête après l'autre. Lorsque l'analyse commandée est terminée, le spécialiste renvoie les résultats à l'agent-donnée concerné.

Cette dichotomie est assez classique en IAD. On la retrouve par exemple dans les plateformes COALA [23] et ETC [47]. Elle offre l'avantage de concilier les modélisations fonctionnelles avec les découpes dirigées par les données.

5.2.1 Agents-données

Dans notre modèle, les agents-données sont chacun responsables d'une partie de l'état courant d'une session. Ils garantissent l'intégrité des informations, maintiennent les liens avec d'autres résultats, et manifestent activement le rôle d'une donnée en commandant différentes analyses aux moments opportuns. La société d'agents-données est dynamique, puisque les agents sont créés à mesure que les résultats sont découverts.

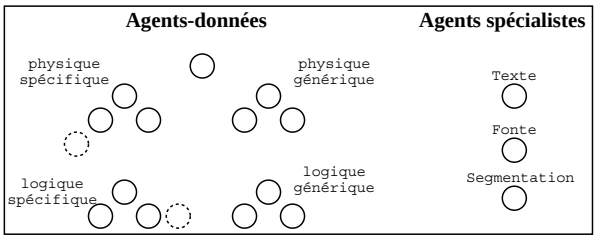


Figure 5.1 : Découpe entre agents-données et spécialistes.

Dans le projet *CIDRE*, il y a de nombreuses manières de partitionner en données actives l'ensemble des informations. Notre proposition respecte l'organisation des données évoquée dans la section 4.2. Chaque entité dominante est contrôlée par un agent. Le contrôle est donc réparti selon les caractéristiques structurelles du document. Le rôle des agents varie suivant les axes spécifique/générique et physique/logique, mais aussi selon le type précis d'entité (bloc, ligne, etc.). Plusieurs arguments justifient cette décentralisation :

- Dans notre organisation des données, les relations hiérarchiques sont prépondérantes; or l'arborescence se révèle une configuration privilégiée pour un système multi-agents, car elle façonne un compromis entre le local et le global.
- En reconnaissance de documents, la remise en cause d'un résultat est un phénomène essentiel dans la construction de la solution. Il nous semble plus élégant d'exprimer localement la détection de conflits, puisque c'est lors de la fusion d'informations contextuelles que les mesures à prendre sont déterminées.
- D'une manière générale, l'enchaînement des analyses est davantage soumis à la présence des données à interpréter qu'au respect d'une algorithmique. Par exemple, un OCR monofonte ne doit pas impérativement s'effectuer *après* la segmentation et la reconnaissance de fonte : conceptuellement, il doit être déclenché dès qu'apparaît une entité textuelle dont la fonte est connue, quelle que soit l'origine de ces informations.

5.2.2 Agents spécialistes

Dans notre décomposition, les agents spécialistes représentent un domaine d'expertise, et se comportent essentiellement comme des serveurs de requêtes. Ils prennent connaissance des besoins du requérant, sondent le contexte des informations disponibles, choisissent les paramètres d'analyse, appellent les outils proprement dits, et convertissent les résultats pour les communiquer aux agents-données. La société de spécialistes est plutôt statique, car les fonctionnalités d'analyse sont en principe connues en phase de programmation.

Il y a plusieurs manières d'organiser en spécialistes l'ensemble des fonctions d'analyse impliquées dans la reconnaissance de documents. Cette conception peut par exemple respecter l'origine des paquetages, même lorsqu'ils offrent des fonctions hétérogènes. Pour notre part, nous avons préféré identifier les *domaines d'interprétation* suivant la nature des informations concernées. Une application *CIDRE* comprendra par exemple des spécialistes pour :

- l'analyse d'image;
- la segmentation;
- la reconnaissance de texte;
- l'identification de fonte;
- l'étiquetage logique.

A priori, rien n'empêche un spécialiste de déléguer une partie de son travail à un autre. Par ailleurs, il peut aussi prendre des initiatives suite au traitement d'une requête : lors de la résolution d'un cas spécial, il pourrait se mettre à la recherche d'une situation similaire ailleurs dans le document, afin de fournir son interprétation sans être explicitement sollicité. Par exemple, après avoir reçu une requête d'apprentissage sur un mot édité par l'utilisateur, le spécialiste d'OCR pourrait parcourir tout le document pour détecter d'autres occurrences mal interprétées.

Plusieurs raisons motivent l'encapsulation de l'expertise au sein d'agents, donc un accès *indirect* aux bibliothèques d'analyseurs :

- Même dans un contexte local, le passage d'un objectif (p. ex. reconnaître le texte) aux moyens (p. ex. pilotage d'un OCR) n'est pas toujours trivial.

- Il est vrai que le choix des méthodes à appliquer dépend en partie de connaissances localisées sur les agents-données. Toutefois, il ne serait pas raisonnable de répliquer dans les différents types d'agents les règles qui déterminent le choix des outils. Par ailleurs, les analyseurs n'ont pas qu'un effet local sur la donnée qu'ils analysent, car c'est la globalité du système qui profite des modifications dues à l'apprentissage incrémental.
- L'interfaçage par domaines d'interprétation permet d'envisager l'extension d'un système par de nouveaux analyseurs avec un minimum d'effort. Si les agents-données appelaient directement les routines d'analyse, le remplacement d'un outil par un autre impliquerait de retoucher le code de plusieurs types d'agents.
- L'analyse d'image est gourmande en temps de calcul. Durant une session typique, les traitements coûteux seront provoqués par le recours aux analyseurs. Or, la responsabilité de l'affectation des ressources de calcul peut difficilement être répartie dans la population d'agents-données; chaque agent-donnée n'a qu'une vue très locale, et ne connaît pas la charge qu'il représente par rapport à l'ensemble du programme. En définissant un nombre limité de spécialistes, on anticipe le problème de la gestion des ressources (cf. section 6.2.3), car toutes les analyses commandées vont transiter par un point de contrôle commun.

5.3 Dichotomie automatique/interactif

Le thème unificateur du projet *CIDRE* concerne une meilleure intégration de l'utilisateur humain dans le processus de reconnaissance. Il nous semble donc que la dimension interactive doit se manifester dès la phase de conception. En effet, le fonctionnement du système est centré sur le rôle de l'opérateur, et l'interface homme-machine est bien plus qu'un détail d'implémentation ou un «sucre esthétique». L'objet de cette section est par ailleurs à rapprocher des modèles d'architecture logicielle issus de l'Interaction Homme-Machine (voir notamment [76]).

Il s'agit donc d'incorporer la gestion du dialogue homme-machine dans notre modélisation multi-agents. Une solution consiste à définir un unique agent responsable de l'interface graphique. Cette approche centralisée s'éloignerait des principes d'IAD, qui suggèrent une granularité plus fine. Une autre solution consiste à organiser des agents suivant la structure des éléments de dialogue, p. ex. un agent par fenêtre. Mais une telle découpe manque d'indépendance par rapport aux choix ergonomiques, puisqu'un changement de présentation (p. ex. transformer des boutons en menus) risque d'engendrer une modification de la découpe en agents.

Nous proposons une troisième voie qui revient à dupliquer la société de base décrite dans la section précédente, comme sur la figure 5.2. Chaque participant, qu'il soit donnée ou spécialiste, se transforme en un couple d'agents : l'*agent d'analyse* gère la dimension automatique et l'*agent de dialogue* est responsable des interactions homme-machine. Cette méthodologie offre plusieurs avantages :

- La responsabilité du dialogue homme-machine, bien que distribuée, est isolée de la partie analytique. On évite ainsi de polluer l'identité des agents par des compétences trop disparates.
- Les interventions de l'utilisateur se propagent au coeur même du système de reconnaissance. Il n'y a pas de goulet d'étranglement entre l'interface graphique et le moteur d'analyse. Conceptuellement, tous les éléments de l'architecture du système sont accessibles à un contrôle manuel.
- Le couplage entre les deux plans est certes formé de liens multiples, mais ces relations sont de type 1:1. L'architecture reste donc très simple, et impose un cadre bienvenu pour spécifier la propagation des événements.

5.3.1 Agents d'analyse

Dans notre découpe du système, les agents d'analyse gèrent la partie automatique de l'environnement assisté. Ils appliquent les méthodes d'analyse ou d'apprentissage, maintiennent l'intégrité des résultats, fusionnent des informations de provenances diverses, détectent les conflits locaux et réagissent en conséquence. En bref, ils utilisent les ressources de calcul pour faire progresser la session.

Les agents d'analyse consacrés aux *données* sont créés et détruits dynamiquement. Leur état interne accumule les informations sur les différentes interprétations subies par la donnée, et les synthétise en une solution courante en tant que membre du document. Ces agents ont une triple mission :

- maintenir la description locale de la structure du document dans un état consistant;

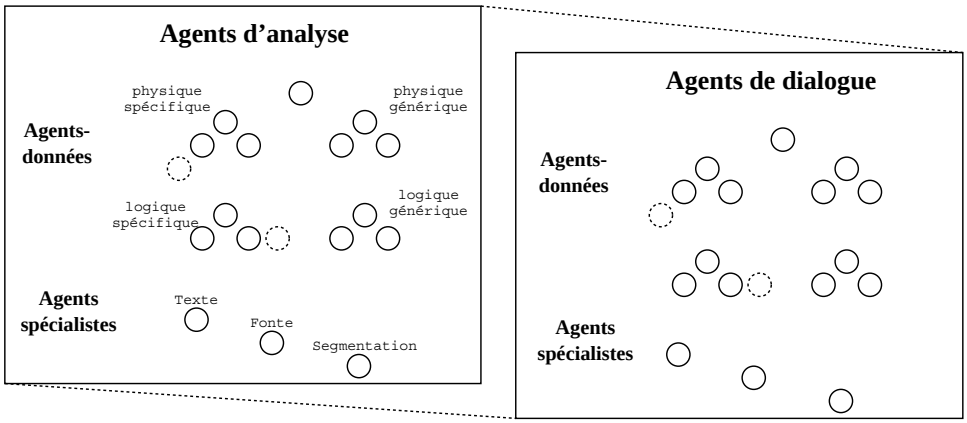


Figure 5.2 : Découpe entre agents d'analyse et agents de dialogue.

- établir les besoins locaux en analyse automatique, à mesure que le contexte de l'entité évolue;
- favoriser la divulgation et l'exploitation des résultats accumulés.

Quant aux agents d'analyse dédiés aux *domaines d'expertise*, ils sont définis une fois pour toutes, et accessibles en tout temps. L'état interne d'un spécialiste décrit les requêtes actuellement en cours de traitement, ainsi que l'expérience acquise, par exemple, par apprentissage. Leur compétence comporte les facettes suivantes :

- effectuer les choix tactiques nécessaires pour servir les requêtes reçues;
- gérer l'affectation des ressources de calcul aux traitements en suspens;
- envisager toutes les possibilités d'améliorer la configuration des analyseurs, en fonction des données à traiter.

5.3.2 Agents de dialogue

Dans notre modélisation, les agents de dialogue gèrent l'interface homme-machine. Ils affichent à l'écran les résultats trouvés, facilitent la navigation dans le document, attirent l'attention sur les parties douteuses, spécifient les commandes autorisées, interceptent les interventions de l'opérateur. D'une manière générale, ils *donnent accès à l'expertise humaine.*

L'état interne des agents de dialogue décrit l'agent d'analyse correspondant en tant qu'objet présenté à l'utilisateur. Cet état évolue *sur le fond* à chaque mise à jour dans l'espace d'analyse, ou *sur la forme* (p. ex changement de vue) sur commande de l'interface utilisateur.

Dans le cas des gestionnaires de *données*, les agents de dialogue s'efforcent de présenter les informations locales de manière visuelle, et d'en faciliter la modification par l'expert humain. Par exemple, les entités physiques spécifiques sont dessinées au-dessus des images de pages, et interceptent les commandes d'édition de texte, de fonte, ou de segmentation.

Dans le cas des *spécialistes*, les agents de dialogue affichent des informations sur les analyses en cours. L'utilisateur pourrait intervenir pour annuler une analyse, en modifier les paramètres, changer l'ordre de priorité, ou allouer plus de ressources.

Notons enfin que le recours à la notion d'agent n'est pas nouvelle dans la conception d'interfaces homme-machine. Le modèle PAC-Amodeus [51], par exemple, organise aussi les éléments de dialogue dans une société d'agents.

5.4 Fonctionnement du système

Les unités de contrôle de notre architecture multi-agents sont des agents-données ou des agents spécialistes, définis dans deux populations symétriques et complémentaires, l'une d'analyse et l'autre de dialogue. L'évolution de ce système est spécifiée par les liens de communication, les types d'interactions, et enfin les comportements individuels de réaction aux événements.

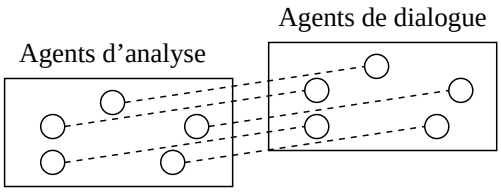


Figure 5.3 : Topologie entre espace d'analyse et interface utilisateur.

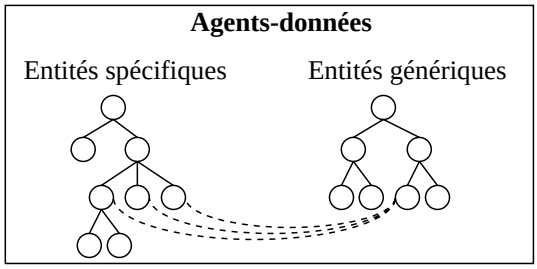


Figure 5.4 : Topologie entre agents-données

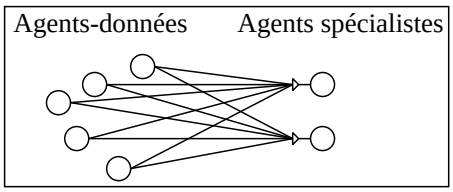


Figure 5.5 : Topologie entre agents-données et spécialistes.

5.4.1 Topologie

La topologie d'un système distribué définit la portée des interactions directes entre correspondants. Dans notre cas, les relations de communication diffèrent selon les catégories d'agents impliquées.

Entre agents d'analyse et agents de dialogue, nous définissons une communication point à point, à l'image de la figure 5.3. Dès leur création, les agents sont associés par couples. Toute la communication entre l'interface utilisateur et le moteur d'analyse est canalisée par ces relations biunivoques.

Entre agents-données, on trouve deux sortes de liens de communication, illustrés sur la figure 5.4. D'une part, les liens de composition ou d'abstraction induisent une communication directe dans les hiérarchies d'entités. Par exemple, un bloc textuel annonce à tous ces fils lignes qu'il a reçu une nouvelle interprétation pour la fonte. D'autre part, les liens de conformité donnent lieu à des flux de communication d'arité 1:n entre entités génériques et spécifiques. Par exemple, l'entité générique «titre» avertit toutes les instances que la règle de syntaxe a changé.

Entre agents-données et agents spécialistes, la communication respecte le protocole clients-serveur suggéré par la figure 5.5. Les spécialistes sont des correspondants accessibles par toute la population d'analyse, et c'est lors de la réception des requêtes qu'ils prennent connaissance des clients.

Entre agents de dialogue, il est difficile de fixer une topologie dominante, car les liens de communication sont dictés par l'application et son mode d'emploi, voire par la puissance de la boîte à outils utilisée. Prenons le cas où un élément de dialogue est accédé de manière partagée par plusieurs agents. La fenêtre affichant la page courante devra p. ex. être accessible par toutes les sous-entités qui désirent dessiner un rectangle pour signaler leur existence. Une solution revient à connecter les agents en étoile : de cette manière, l'agent page devient responsable de gérer la fenêtre, et maintient un lien direct vers tous les agents qui cherchent à interagir avec cet élément de dialogue. La communication peut parfois être véhiculée par le gestionnaire d'événements lui-même, pour autant que celui-ci offre un mécanisme d'*abonnement* à un widget, ou de *tags* comme dans Tcl-Tk [165].

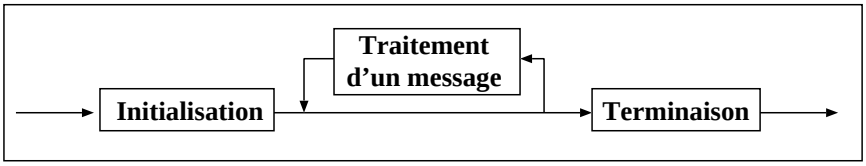


Figure 5.6 : Schéma d'exécution général d'un agent.

5.4.2 Interactions

Pour spécifier les interactions entre agents, nous adoptons un modèle de *messagerie*. L'acheminement d'un message déclenche l'activation de l'agent destinataire. Les messages sont typés, et l'interface d'une classe d'agents détermine l'ensemble des messages acceptés.

Dans l'espace d'analyse, les messages reçus par les agents-données peuvent refléter les connotations suivantes :

- *modifier* : une nouvelle valeur est proposée pour un attribut de la donnée maintenue;
- *prendre note* : une information présente dans le contexte de la donnée a été mise à jour;
- *remettre en cause* : une partie des résultats attachés à l'entité est critiquée.

Les spécialistes sont activés pour l'un des motifs suivants :

- *analyser* : l'analyse d'une certaine portion des données est commandée;
- *apprendre* : le spécialiste est appelé à enrichir ses connaissances par apprentissage sur des résultats tenus pour corrects;
- *réallouer* : la priorité de certains traitements en cours d'exécution est modifiée;
- *configurer* : on met à jour les paramètres du spécialiste, p. ex. l'outil d'analyse par défaut, ou le répertoire temporaire.

Quant aux agents de dialogue, les messages en provenance de l'espace d'analyse concernent surtout le rafraîchissement des informations présentées à l'utilisateur. Les interactions à l'intérieur de l'espace de dialogue concernent typiquement la création d'éléments graphiques, la gestion de la sélection, les modes de visualisation, ou la mise en oeuvre d'une fonctionnalité non triviale comme rechercher-remplacer.

D'une manière générale, il est difficile d'énumérer plus précisément les types de messages sans fixer auparavant le cahier des charges de l'application visée. Pour notre part, nous nous sommes concentrés sur les problèmes de reconnaissance de structure physique. On trouve donc des cas particuliers d'interactions entre agents dans la discussion sur le prototype (cf. section 6.4), et sur les méthodes d'analyse (cf. chapitre 7).

5.4.3 Règles de comportement

Fonctionnement général

Dans notre système, les comportements individuels sont définis par des règles de réaction aux événements perçus. L'initialisation et la terminaison d'un agent sont deux événements particuliers, auxquels s'ajoute la réception d'un message. Comme esquissé sur la figure 5.6, le schéma d'exécution d'un agent est donc très simple. Le traitement d'un événement dépend de plusieurs éléments :

- le type d'agent;
- l'état interne de l'agent au moment de la réception;
- le type, le contenu, voire l'expéditeur du message.

Le traitement d'un événement fait évoluer le système grâce à deux effets, à savoir la modification de l'état interne de l'agent activé, et le déclenchement de nouvelles interactions par envoi de messages. Le concepteur d'une application basée sur notre architecture devra spécifier, pour chaque type d'agents prévu et chaque type de messages possible, l'algorithme qui décidera : (i) comment adapter les données locales, et (ii) quels sont les conséquences du message reçu sur chacun des correspondants de l'agent.

Exemple

La figure 5.7 illustre en pseudo-code le comportement d'un agent d'analyse gérant une ligne de texte. L'extrait du scénario que nous discutons est schématisé sur la figure 5.8. Suite à la réception d'un message (i) apportant

```
PROC LineAgent (self: ThreadContext);
VAR inMsg, outMsg: Message;
    myLine:        DafsEntity;
    ocrSpecialist: Correspondant
    ...
BEGIN
  ...
  ocrSpecialist = self.private.ocr;
  myLine = self.private.dafsEntity;
  StartTwin();
  ...
  LOOP
    ReceiveMsg (inMsg);                                  (i)
    CASE inMsg.tag OF
      ...
    modifyFontMSG:
      IF IsFontConflict(myLine, inMsg.font) THEN
        ...
      ELSE
        SetFontProp(myLine, inMsg.font);                 (ii)
        outMsg.tag = notifyMSG;
        SendUp(outMsg);                                  (iii)
        BroadcastDown(outMsg);                           (iv)
        SendToGeneric(outMsg);                           (v)
        SendToTwin(outMsg);                              (vi)
        outMsg.tag = anslyseREQ;
        SentMsg(ocrSpecialist, outMSG);                  (vii)
      END IF;
    ...
    END CASE;
  END LOOP;
END LineAgent;
```

Figure 5.7 : Pseudo-code d'un agent-donnée du type ligne

une interprétation typographique, l'agent, pour qui cette valeur ne provoque pas de conflits, accepte de modifier l'attribut fonte (ii). Il en avertit son parent du type «bloc» (iii), ses fils du type «mot» (iv), et son modèle du type «ligne générique» (v). Il commande à son agent de dialogue de modifier la présentation en conséquence (vi) : si une interprétation du texte existait déjà, l'affichage sera modifié avec la bonne fonte; sinon, on peut mémoriser l'attribut, au cas où l'utilisateur demande à consulter la fonte. Enfin, notre agent d'analyse détecte que ce nouveau résultat le rend disponible pour une reconnaissance de texte, et lance une requête vers le spécialiste concerné (vii).

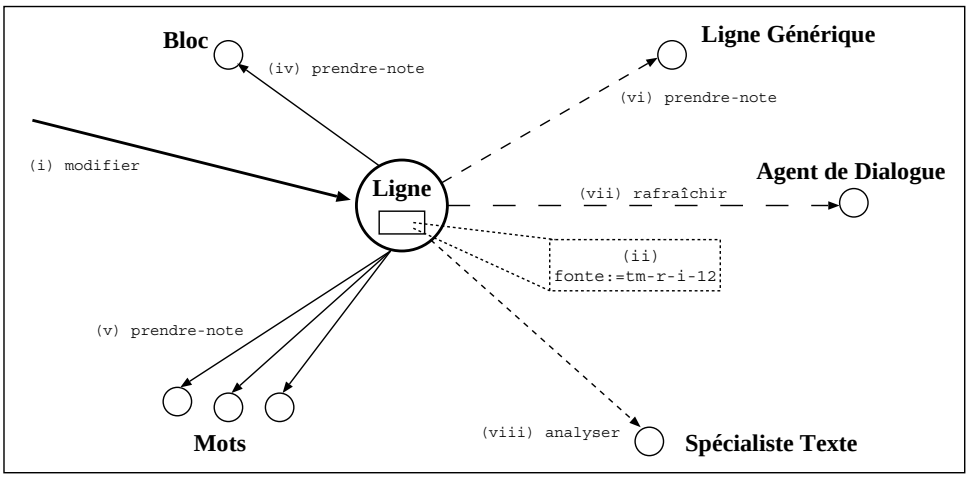


Figure 5.8 : Conséquences d'un événement pour un agent-donnée.

Comportements inspirés de l'IAD

Notre architecture permet d'exprimer des comportements très variés, des plus simples aux plus complexes. Tout dépend des objectifs de l'application. Nous ne possédons pas de recette universelle pour concevoir la stratégie adéquate. Il s'agit de réussir le passage entre une vision globale du comportement souhaité et l'expression des comportements locaux. Il faut aussi faire preuve d'imagination et trouver des astuces qui simulent des réactions «intelligentes».

Si on désire respecter au maximum l'approche multi-agents, il faudrait que les comportements individuels exhibent les deux phénomènes suivants :

- les conséquences d'un événement dépendent effectivement de l'état interne de l'agent;
- l'état interne reflète l'historique des interactions subies par l'agent.

Si la première condition n'est pas satisfaite, cela signifie que le contrôle n'est pas vraiment décentralisé, puisqu'il n'y a pas de prise de décision qui porte sur des critères locaux. Si la deuxième condition manque, l'agent n'évolue pas vraiment, puisqu'il ne garde aucune trace des événements qui traversent son existence. Lorsque les deux conditions sont remplies, le comportement du système devient intéressant pour l'IAD, car le rôle des individus est alors conditionné par l'évolution de la société, elle-même indissociable des comportements individuels.

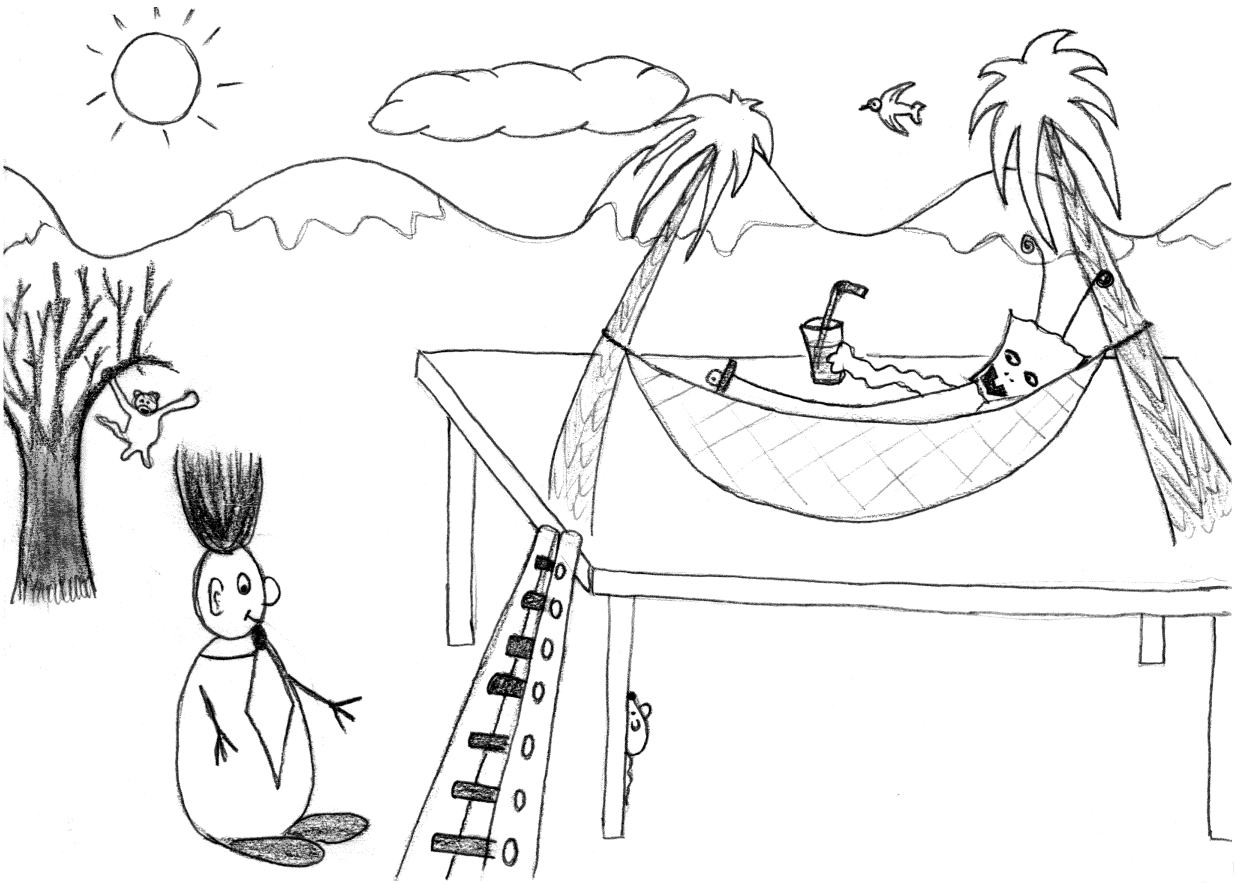
Conclusion

Nous avons décomposé le système de reconnaissance en une population d'agents semi-autonomes, doués d'interaction. Notre modélisation est basée sur une découpe selon les données et les tâches, ainsi que sur la séparation entre traitements automatiques et gestion du dialogue. Notre approche intègre l'utilisateur humain comme un participant privilégié, tient compte du parallélisme inhérent au traitement des documents, et ne force pas le développeur à figer chaque détail du schéma d'analyse.

Le prochain chapitre présente une plateforme d'implémentation qui permet de concrétiser notre architecture conçue comme système multi-agents.

Chapitre 6

Plateforme d'implémentation



L'architecture abstraite présentée dans le chapitre précédent doit maintenant être concrétisée. Il y aurait de nombreuses approches pour simuler notre système multi-agents. Ce chapitre présente les caractéristiques de notre implémentation.

Nous commençons par énumérer les propriétés que devrait offrir l'environnement de programmation afin de faciliter la mise en oeuvre du système. La section 6.2 présente la plateforme utilisée pour programmer les agents du moteur d'analyse. La section 6.3 discute la mise en oeuvre de l'interface graphique, et explique comment réaliser le couplage avec le noyau analytique. La dernière section (6.4) décrit le prototype qui a servi à tester notre plateforme logicielle.

6.1 Support de programmation requis

La mise en oeuvre d'une application basée sur la philosophie *CIDRE* représente un gros travail de développement, qui demande une bonne formation de programmeur. En outre, le choix de la plateforme de développement est déterminant pour la réussite du projet. L'approche multi-agents pose les bases de l'architecture, mais il y aurait de très nombreuses manières de la mettre en oeuvre.

En choisissant l'IAD comme source d'inspiration, nous avons envisagé de suivre un modèle multi-agents

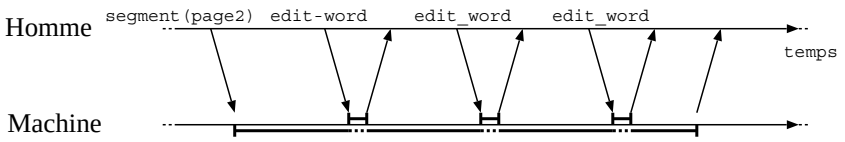


Figure 6.1 : Dialogue homme-machine asynchrone.

déjà établi et les outils logiciels associés [121, 23, 47, 122, 183]. Toutefois, la discipline est encore jeune, les plateformes ne sont pas toujours destinées à la résolution de problèmes, et l'analyse d'images de documents nous impose certaines contraintes, p. ex. sur le langage de programmation.

Nous estimons que notre modélisation multi-agents est essentiellement une abstraction, qui peut être simulée par des langages de programmation conventionnels. L'annexe A évoque divers mécanismes qui permettent de simuler la notion d'agent.

Nous allons discuter les principaux supports que doit offrir l'environnement de programmation, en vue (i) de respecter l'approche multi-agents, et (ii) de tenir compte des autres contraintes du projet *CIDRE*. A notre avis, la plateforme logicielle présentée plus loin répond aux critères présentés ici.

6.1.1 Exécution asynchrone

Deux raisons majeures nous incitent à choisir un moteur d'exécution asynchrone : le rôle central de l'utilisateur, et le respect de l'approche multi-agents. Le concept de reconnaissance assistée revalorise le rôle de l'opérateur humain. La convivialité impose que le pilotage de la session ne soit jamais bloqué. Or, les algorithmes utilisés en reconnaissance de documents engendrent des temps de calcul qui sont loin d'être négligeables. Il est donc important que l'utilisateur puisse continuer à travailler durant l'exécution de ces traitements. Le côté asynchrone est un élément naturel dans un dialogue coopératif (cf. section 3.4). Sur la figure 6.1 par exemple, l'utilisateur poursuit la correction des résultats d'OCR sur une page pendant la segmentation automatique de la page suivante.

Non seulement l'interface graphique doit être prête à intercepter les interventions de l'utilisateur, mais le moteur d'analyse doit réagir en conséquence, même si un traitement coûteux n'est pas encore terminé. Pour permettre un tel comportement de l'application, notre architecture devrait supporter l'exécution de traitements concurrents.

Notre modélisation, abstraite, admet un fonctionnement autonome des agents, et l'absence de formes cachées de synchronisation. Par contre, des contraintes de synchronisation risquent d'être induites par l'environnement d'implémentation (cf. section 6.3.2 et annexe A). Par exemple, si on cherche à traduire l'architecture multi-agents en C++, la notion d'envoi de message s'exprime par un invocation de méthode. Dans ce cas, les interactions sont dénaturées au profit d'une exécution séquentielle et synchrone, puisque l'agent expéditeur ne poursuit son exécution qu'une fois que le destinataire aura complètement traité le message. A notre avis, s'éloigner du modèle abstrait serait préjudiciable à plus d'un titre. Premièrement, le programmeur conscient de ces phénomènes risque de les accepter comme hypothèses de travail, en abandonnant de fait la vision locale. Deuxièmement, même si le programmeur se discipline à expliciter les dépendances entre agents, le comportement du système reste évidemment soumis au mécanisme d'activation du langage. Par rapport à ce que spécifie le modèle multi-agents, le programme va donc privilégier systématiquement les mêmes enchaînements de procédures. Une évolution synchrone appauvrit sensiblement l'environnement local des agents. L'introduction de ce biais est également néfaste pour l'utilisateur, qui perçoit un moteur d'analyse très répétitif.

6.1.2 Exploitation du parallélisme

En reconnaissance de documents, beaucoup de techniques d'analyse sont gourmandes en temps de calcul, et les contraintes sur les temps de réponse sont difficiles à respecter. L'optimisation des performances est donc un objectif important, dont les implications sur les choix du développeur sont multiples : matériel, techniques d'analyse, résolution des images, etc.

Du point de vue de l'architecture, nous comptons en particulier sur le calcul distribué [200] pour améliorer les performances (cf. sect. 3.6.2). D'ailleurs, la tendance générale dans l'évolution du matériel s'oriente vers les machines multi-processeurs. Il nous semble important que la plateforme d'implémentation soit capable d'exploiter toute la puissance de calcul disponible, qu'elle soit distribuée sur un réseau de stations de travail

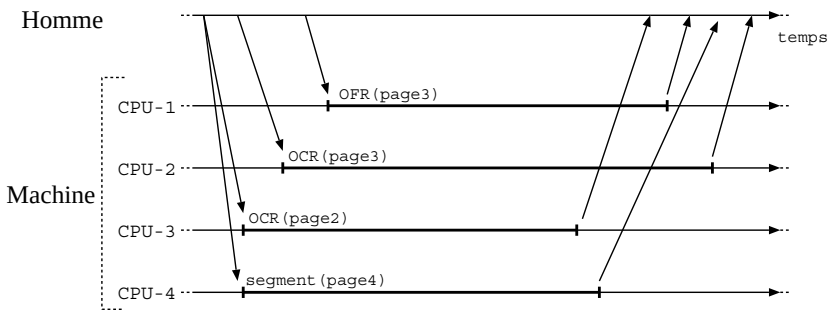


Figure 6.2 : Parallélisation des traitements.

ou dans un super-calculateur.

Grossièrement, on peut distinguer deux approches pour la gestion du parallélisme :

- parallélisation *implicite* : le programmeur écrit un code séquentiel, et c'est le compilateur (voire le processeur) qui se charge de paralléliser au mieux le flux d'instructions.
- parallélisation *explicite* : le programmeur identifie les parties du code parallélisables, et exprime les relations de synchronisation.

Seule l'approche explicite a des conséquences sur l'architecture logicielle. En reconnaissance de documents, on peut aborder le parallélisme de plusieurs manières. Notons d'abord qu'il existe des algorithmes parallèles de traitement d'images [179]. Dans de nombreuses applications comme le tri postal, l'unité de parallélisme la plus raisonnable est le document lui-même. Dans l'approche *CIDRE*, nous comptons sur le parallélisme pour accélérer les analyses automatiques, afin de diminuer les temps d'attente dans une session de travail interactive. Notre plateforme devrait surtout permettre d'exprimer une exécution distribuée au niveau de l'appel des routines d'analyse. Le parallélisme inhérent à notre application provient de deux caractéristiques du problème :

- *indépendance des données :* par exemple, deux pages peuvent être segmentées simultanément;
- *indépendance des tâches :* par exemple, une même ligne de texte peut être analysée simultanément par un OCR et par une reconnaissance de fonte.

La figure 6.2 illustre le fonctionnement d'un système où les traitements automatiques sont exécutés en parallèle sur un réseau d'ordinateurs. Du moment qu'on admet un fonctionnement asynchrone, la répartition physique des tâches n'a pas d'influence sur la conduite du dialogue homme-machine. En revanche, on peut espérer que les résultats apparaîtront plus rapidement que dans un moteur d'analyse non distribué.

6.1.3 Extensibilité

Nous estimons qu'un système de reconnaissance de documents est par nature ouvert aux modifications et aux extensions :

- aucun analyseur n'est parfait, et on peut toujours chercher une méthode plus rapide, plus précise, ou plus générale;
- la communauté scientifique propose en permanence de nouveaux algorithmes, qui ont chacun leurs avantages;
- la nature des données a une grande influence sur les résultats, et une légère modification des conditions d'utilisation peut engendrer des baisses de performance sensibles; si on veut utiliser le logiciel dans un contexte un peu différent, il est parfois nécessaire d'ajouter du code.

Par conséquent, notre architecture doit prévoir l'intégration de nouveaux outils d'analyse. Notre gestion uniforme des données basée sur DAFS est un premier pas dans ce sens. Au niveau de la plateforme de programmation, le souci d'extensibilité requiert la compatibilité avec les environnements les plus utilisés en analyse d'images de documents.

Toujours concernant la maintenance du système de reconnaissance, il faut naturellement être attentif aux problèmes de portabilité des sources.

Concrètement, l'environnement de programmation doit offrir un support pour utiliser des librairies C ou C++, puisque la majorité des analyseurs disponibles sont écrits dans ces langages. Il devrait aussi prévoir

l'utilisation d'outils qui ne sont disponibles que sous forme d'exécutables; sous Unix par exemple, l'application pourrait les exploiter en gérant plusieurs processus lourds.

Il y a enfin une décision importante et incontournable, qui concerne justement le système d'exploitation visé. A l'heure actuelle, les alternatives se limitent en pratique à Unix, Windows, ou MacOS. Dans notre cas, le choix s'est immédiatement porté sur Unix, étant donné que notre équipement est essentiellement formé de stations de travail Sun, installées avec le système Solaris. Il nous semble que c'est toujours l'environnement le plus utilisé dans le milieu académique, bien qu'on observe une tendance à migrer vers Windows.

6.1.4 Rapide prototypage

Le confort de l'utilisateur joue un rôle primordial dans l'approche *CIDRE*. En pratique, cela signifie que le cahier des charges idéal ne sera déterminé qu'après avoir procédé à l'évaluation de différentes versions du logiciel dans le terrain. La plateforme d'implémentation doit offrir un support pour construire des prototypes complets dans de brefs délais, de manière à pouvoir tester plusieurs variantes à peu de frais. Après avoir expérimenté une série de pré-versions, on est mieux armé pour élaborer le produit définitif. C'est surtout la mise en oeuvre de l'interface homme-machine qui risque de faire l'objet de nombreuses retouches, parfois jusqu'en profondeur.

Pour parer à ces problèmes de développement, plusieurs solutions, d'ailleurs complémentaires, sont envisageables :

- utiliser un toolkit graphique de haut niveau, qui offre la possibilité de structurer le code de l'interface homme-machine, par exemple en définissant de nouvelles classes d'élément de dialogue par agglomération d'éléments existants;
- recourir à un générateur d'interface, qui libère le programmeur du codage proprement dit de la présentation graphique;
- programmer l'interface graphique à l'aide d'un langage interprété, afin d'accélérer les mises à jour en évitant la phase de compilation.

6.2 Programmation concurrente et distribuée avec PT-PVM

Pour piloter le moteur d'analyse de notre architecture, nous avons choisi la plateforme de développement PT-PVM, mise au point dans le groupe Parallélisme de notre institut [112].

6.2.1 PT-PVM

PVM [196, 74] (Parallel Virtual Machine) est une plateforme logicielle qui permet de simuler une machine parallèle à partir d'un ensemble de stations de travail connectées par un réseau local. Le modèle de programmation de PVM est basé sur une messagerie asynchrone. Les unités de parallélisme sont formées de tâches avec espace d'adressage disjoint, typiquement implémentées sous forme de processus lourds Unix. Le paquetage PVM offre une librairie en C et des utilitaires, p. ex. pour visualiser le déroulement du programme distribué (XPVM). En pratique, PVM est la plateforme de développement la plus utilisée par les spécialistes en programmation distribuée. Le concept de machine virtuelle, mis en oeuvre par PVM, apporte les avantages suivants :

- *configuration dynamique :* des machines peuvent être ajoutées durant l'exécution;
- *parc hétérogène :* plusieurs types de machines peuvent être intégrées simultanément;
- *indépendance physique :* les unités de traitement sont nommées de manière uniforme quelle que soit la machine hôte;
- *efficacité :* la mise en oeuvre utilise un minimum de ressources pour gérer le fonctionnement du système.

PT-PVM [114, 112] est une extension à PVM, proposée et mise en oeuvre par le groupe Parallélisme de notre institut. Cette plateforme logicielle apporte des solutions originales et concrètes aux difficultés soulevées par la programmation distribuée. Nous allons mettre ici l'accent sur deux innovations introduites par PT-PVM : (i) le support de threads au sein de PVM, et (ii) l'intégration de processus Unix quelconques (c.-à-d. qui n'ont pas été créés avec PVM) dans une application distribuée.

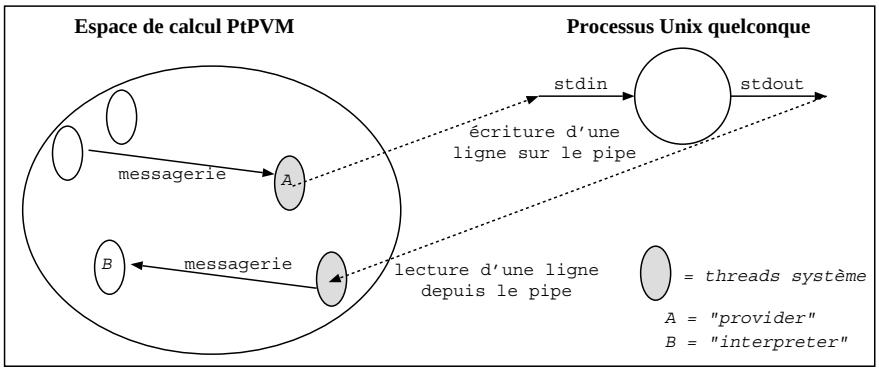


Figure 6.3 : Processus UNIX intégré dans PT-PVM.

Programmation multi-threads massivement parallèle

PT-PVM étend la puissance de PVM en affinant la granularité. Rappelons qu'en systèmes d'exploitation, on parle de processus lourd ou léger selon l'étendue des informations qui leur sont associées. Le changement de contexte entre processus légers est plus rapide car ils se partagent certaines ressources, comme l'espace d'adressage ou les descripteurs de fichiers. La plateforme PT-PVM offre un support complet pour gérer des *processus légers préemptifs* (ou *filets d'exécution* ou *threads*) [159, 181, 125] dans une application distribuée [77]. Chaque thread est identifié univoquement dans l'espace distribué en tant que *correspondant* pour les opérations de messagerie. Dans cet environnement multi-threads, les filets d'exécution sont attachés à un *environnement de processus*, qui fait office d'hôte lourd. A l'intérieur d'un environnement de processus, les filets se partagent le même espace mémoire.

Cette approche mixte est destinée à simuler le fonctionnement de systèmes massivement parallèles. En effet, les systèmes d'exploitation imposent des limites aussi bien sur le nombre de processus lourds par machine (et par utilisateur), que sur le nombre de threads par processus lourd. Le recours exclusif à l'un ou l'autre des mécanismes de concurrence convient aux applications qui doivent gérer une petite population de processus, de l'ordre de la centaine. En combinant des processus lourds et légers dans un environnement distribué, PT-PVM permet d'envisager la réalisation de systèmes beaucoup plus fragmentés, et se prête bien à la simulation de systèmes multi-agents.

Pour notre application, ce changement d'ordre de grandeur est important : nous devons gérer une forte population d'agents, puisque dans notre décomposition orientée selon les données, chaque entité du document engendre la création d'un agent. Concrètement, nous pouvons implémenter chaque agent-donnée par un filet d'exécution, en conservant l'asynchronisme du modèle multi-agents.

Interopérabilité

PT-PVM permet d'interagir de manière consistante avec des processus UNIX démarrés depuis n'importe quel programme exécutable. Cette deuxième propriété accomplit un pas de plus vers l'interopérabilité, puisqu'il n'est plus obligatoire de posséder les sources des programmes pour les intégrer dans l'espace de calcul distribué.

L'idée consiste à considérer que tout processus Unix possède un flux de messagerie par l'intermédiaire de l'entrée et de la sortie standards (stdin, stdout). Par conséquent, il suffit de mettre en oeuvre une couche qui interface cette forme de communication de manière consistante avec le modèle de messagerie de PT-PVM. Le mécanisme repose sur deux threads créés par le système : le premier représente le processus Unix en tant que destinataire de messages, et réécrit sur le pipeline tout ce qu'il reçoit ; le deuxième joue le rôle d'expéditeur, et redirige ce qu'il lit sur le pipeline vers un correspondant convenu à l'avance. En somme, le pilotage du processus Unix à travers le pipeline s'effectue de manière transparente, puisqu'une fois le processus créé, le reste de l'application distribuée interagit avec lui comme avec n'importe quel autre correspondant.

La figure 6.3 illustre le rôle du processus Unix par rapport à l'espace de calcul PT-PVM. La figure 6.4 présente une spécification de la primitive, offerte par PT-PVM, qui permet de générer un processus Unix et de l'intégrer dans l'espace de coordination.

Il est intéressant de noter que c'est durant l'implémentation de notre architecture de reconnaissance de documents que nous avons proposé ce mécanisme, qui se révèle une extension très expressive de PT-PVM.

```
PROC SpawnUnix(            -Routine de création dédiée aux processus Unix
IN          cmd:  String   -Commande Unix à exécuter
IN  interpreter:  Corresp  -Correspondant destinataire des lignes lues sur stdout
IN          tag:  Integer  -Etiquette des messages expédiés par le processus Unix
IN       onHost:  String   -Machine du réseau qui exécutera le processus Unix
OUT    provider:  Corresp) -Correspondant qui redirigera tous les messages reçus
                            vers stdin
```

Figure 6.4 : Génération d'un processus UNIX quelconque avec la librairie PT-PVM.

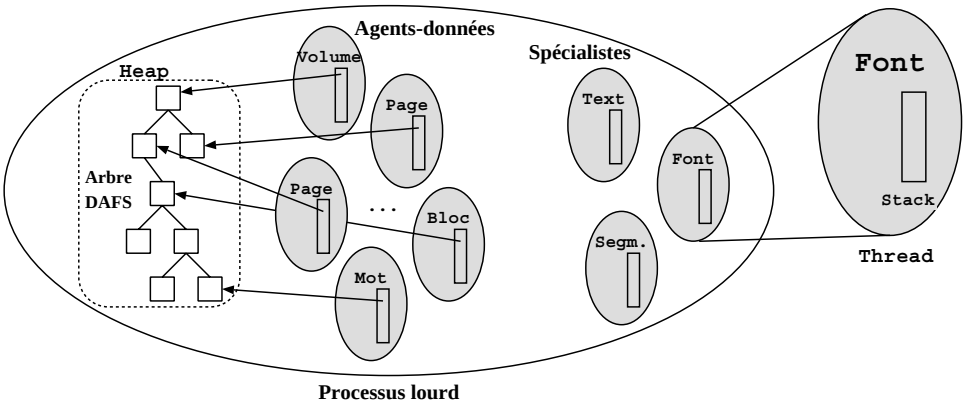


Figure 6.5 : Implémentation des agents d'analyse par un ensemble de threads.

L'utilisation de PT-PVM dans l'architecture du projet *CIDRE* a été une des premières applications de la plateforme. Pour nous, il s'agissait de traduire une partie de notre modélisation multi-agents (les agents d'analyse) avec les concepts de coordination de PT-PVM. Depuis, PT-PVM a également été choisi comme base pour réaliser des expériences avancées sur les systèmes d'agents autonomes [115, 40].

6.2.2 Processus légers et mémoire partagée avec PT-PVM

Notre moteur d'analyse est modélisé par un système multi-agents. La réalisation que nous proposons consiste à concrétiser chaque agent par un processus PT-PVM. Il s'agit donc d'organiser l'espace de coordination PT-PVM pour inclure des agents-données et des spécialistes. Toute la population travaille en commun à enrichir une solution courante, représentée au format DAFS. Dans un environnement multi-thread [159], les filets d'exécution créés dans un même processus lourd se partagent l'espace d'adressage. Cette propriété va nous permettre de contourner deux obstacles :

- tout un arbre de données DAFS peut être stocké en un seul morceau dans la mémoire, malgré un contrôle décentralisé du code qui en gère les différents noeuds;
- la messagerie peut être court-circuitée par des accès directs aux données gérées dans le voisinage de l'agent. On évite alors de surcharger le système avec des échanges de messages peu porteurs, puisqu'ils ne servent qu'à lire des résultats dans le contexte local.

La figure 6.5 illustre une première implémentation de la société d'agents d'analyse avec la plateforme PT-PVM. Par souci de clarté, nous nous concentrons sur la structure physique spécifique. Dans cette variante, que nous avons implémentée, tous les agents-données s'exécutent dans un unique environnement de processus, et se partagent la gestion de l'arbre des résultats DAFS de manière concurrente. Les spécialistes sont aussi des threads placés dans le même processus lourd, afin d'avoir un accès direct à toute la solution courante.

Cette première mise en oeuvre suffit pour programmer un prototype rudimentaire, mais la capacité limitée de l'unique processus lourd empêche de gérer des gros documents multi-pages. Les ressources mémoire allouées au processus lourd seront insuffisantes pour créer tous les agents données, surtout si plusieurs images sont chargées. Pour supporter le changement d'échelle, il faudrait fragmenter le contrôle en plusieurs processus lourds. La figure 6.6 montre une nouvelle mise en oeuvre adaptée aux gros volumes de données. Dans cette proposition, que nous n'avons pas expérimentée, l'arbre global est sectionné au niveau de la page. Tous les agents gérant le descendant d'un noeud page forment un processus lourd. Quant aux spécialistes, ils sont regroupés dans un seul environnement de processus. Cette découpe accroît les communications dans le système, mais offre les avantages suivants :

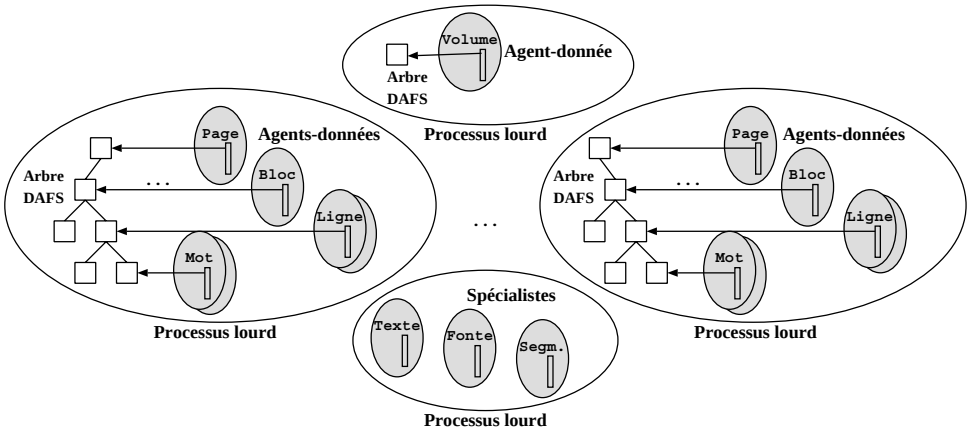


Figure 6.6 : Implémentation des agents d'analyse par plusieurs ensembles de threads.

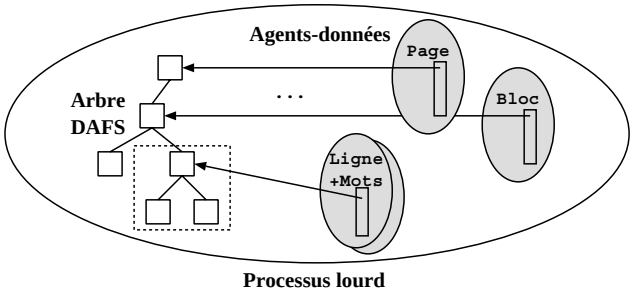


Figure 6.7 : Simulation du niveau mot par les threads gérant les lignes.

- on obtient le bon ordre de grandeur, aussi bien sur le nombre de threads par processus (env. un millier), que sur le nombre de processus lourds dans tout l'espace distribué (env. une centaine);
- les images, qui sont les unités d'information les plus volumineuses, sont chacune chargées en mémoire seulement deux fois, dans le processus qui contient le sous-arbre page et dans celui dédié aux spécialistes. Dans d'autres découpes en processus lourds, on risquerait encore plus de redondance.

Nous proposons encore un autre moyen de réduire le nombre de threads : il consiste à simuler de manière non concurrente le dernier niveau de découpe. La figure 6.7 illustre une mise en oeuvre où les agents mots sont en fait encapsulés dans le thread gérant la ligne. Le processus associé à une ligne réceptionne tous les messages qui sont conceptuellement adressés aux agents-mots, et exécute lui-même la réaction nécessaire. Un tel mécanisme, que nous avons utilisé dans notre prototype, introduit des obstacles dans l'expressivité du comportement des individus, même si le biais par rapport au modèle conceptuel d'agent reste léger.

6.2.3 Processus lourds et configuration en réseau

Pour le projet *CIDRE*, les capacités de programmation distribuée sont surtout utiles pour accélérer l'exécution des requêtes d'analyse. En effet, parmi toutes les unités de contrôle de notre système multi-agents, on s'attend à ce que les spécialistes consomment la majeure partie des ressources de calcul.

Dans notre architecture, les relations entre agents-données et agents spécialistes sont du type clients-serveurs. En l'occurrence, les requêtes se caractérisent par une durée de traitement relativement longue. Nous proposons d'introduire un niveau de contrôle supplémentaire, afin de séparer proprement d'une part le rôle du spécialiste et d'autre part l'exécution des procédures d'analyse. Chaque serveur pilote un ensemble de *travailleurs* qui ne font qu'appeller les analyseurs.

Ce paradigme clients-serveurs-travailleurs permet d'organiser le contrôle en répartissant la charge de calcul sur les processeurs disponibles. La figure 6.8 illustre un système organisé selon ce schéma, et qui s'exécute sur un réseau de machines. Le serveur délègue les requêtes coûteuses aux travailleurs, qui sont placés sur différents processeurs. De cette manière, la gestion du parallélisme est transparente pour les clients, dans notre cas, les agents-données. De plus, le programme reste indépendant de la machine hôte, et traduit la distinction entre stratégie d'analyse et allocation des ressources. Par exemple, on peut facilement modifier le nombre de travailleurs ou la configuration du réseau, ou encore mettre en oeuvre des techniques pour

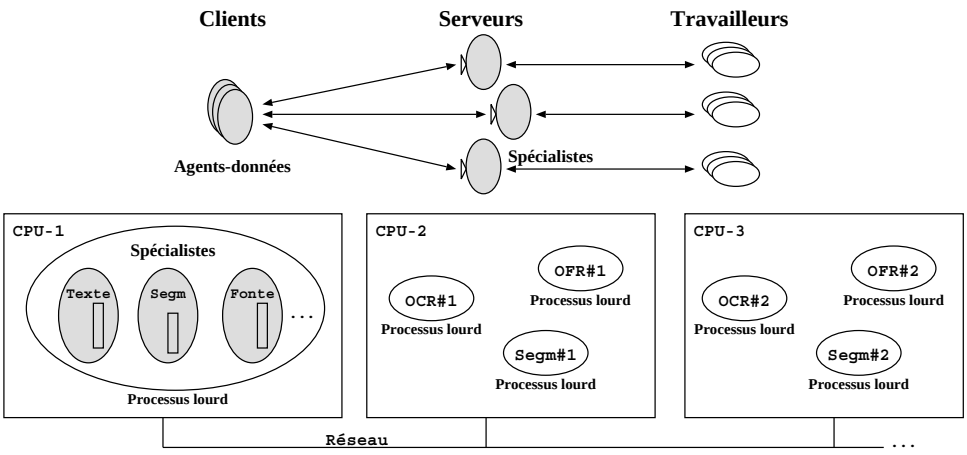


Figure 6.8 : Répartition des travailleurs sur le réseau.

équilibrer les taux de charge [195, 190].

Si nous n'avons pas introduit la notion de travailleur dans la conception multi-agents, c'est que ces unités de contrôle ne servent qu'à accélérer le débit des requêtes servies par les spécialistes. Les travailleurs fonctionnent comme des esclaves dénués d'autonomie ou d'adaptabilité. La délégation de traitements à un travailleur alourdit certes le schéma d'exécution, mais se justifie par les avantages suivants :

- le spécialiste est davantage disponible pour recevoir les requêtes des clients, et on diminue ainsi le risque d'un goulet d'étranglement;
- le détail des appels aux analyseurs est isolé du code du spécialiste, qui se concentre sur un niveau d'abstraction un peu plus élevé;
- le problème de l'allocation des ressources de calcul est isolé du reste de l'application, ce qui va permettre d'exploiter le CPU le plus judicieusement possible.

6.3 Programmation multi-langages avec C et Tcl-Tk

La philosophie *CIDRE* accorde une place prépondérante à l'ergonomie. L'architecture logicielle doit donc tenir compte des difficultés liées au développement d'une interface graphique conviviale. Dans notre plateforme d'implémentation, nous proposons d'utiliser le langage Tcl [165] pour programmer l'interface utilisateur. Nous commençons par exposer les motivations qui nous incitent à mélanger les langages de programmation dans notre architecture. Ensuite, nous proposons une implémentation en Tcl de la société d'agents de dialogue. La dernière section soulève la question du couplage avec les agents d'analyse, comme un cas particulier de couplage entre langages de prgrammation.

6.3.1 Approche multi-langages

Le terme *multi-langages* fait référence au mélange, au sein d'un même logiciel, de sources écrits dans différents langages de programmation. Il s'agit d'un cas particulier d'*interopérabilité*. Le besoin de combiner entre eux des langages de programmation part d'un constat très simple : chaque environnement de programmation a ses propres avantages et inconvénients. La science informatique dispose de différents paradigmes de programmation : instructions impératives, règles déclaratives, objets, contraintes, script, concurrence, événements, flux de données, et bien d'autres. Chacun se révèle plus ou moins bien adapté pour implémenter telle classe d'algorithmes ou pour modéliser tel problème. Or une application en vraie grandeur a souvent à gérer des sous-problèmes variés, et il est tentant d'utiliser, pour chaque partie, le langage le plus approprié. L'interopérabilité résout aussi le problème des échanges entre programmes dans un environnement hétérogène, typiquement dans des applications distribuées sur le réseau. Parmi les architectures standardisées qui apportent une réponse à ce problème, Corba [49] est le représentant le plus connu.

L'approche multi-langages apporte les avantages suivants :

- *Expressivité :* le programmeur peut utiliser le bon niveau d'expression pour chaque partie du logiciel.
- *Modularité :* la frontière des langages force le développeur à définir clairement les compétences de

chaque module.

- *Réutilisabilité :* les paquetages qui ont fait leur preuve sont réutilisés sans devoir les réécrire de manière artificielle dans un nouveau langage.
- *Optimisation locale :* puisque les différents modules sont chacun écrit dans le langage approprié, ils peuvent être optimisés individuellement, que ce soit en lisibilité ou en performance.

Il faut néanmoins tenir compte de plusieurs inconvénients:

- *Gestion des dépendances :* L'environnement de développement ne peut plus assister le programmeur dans la gestion des dépendances entre modules. Il faut alors se discipliner pour maintenir l'intégrité dans le projet global.
- *Optimisation globale :* la frontière des langages contraint les interactions entre modules, et l'optimisation des composants ne provoque pas forcément l'optimisation du programme global, notamment en terme d'expressivité.
- *Compétences supplémentaires :* l'équipe de développement ou de maintenance doit maîtriser plusieurs langages de programmation.

Dans notre projet *CIDRE*, c'est la dichotomie automatique-interactif qui remet en cause une réalisation mono-langage. En effet, nous avons choisi une plateforme basée sur C et Unix pour implémenter les fonctionnalités automatiques, parce que c'est l'environnement de programmation naturel de nos analyseurs d'images de documents. Or, nous prétendons que l'environnement C/Xlib [161] n'est pas idéal pour programmer une interface graphique complexe. Voici quelques points que nous lui reprochons :

- La programmation des différents toolkits (Motif [78], OpenWindows) est difficile à maîtriser. Les structures de données sont généralement complexes.
- Les opérations de compilation ralentissent sensiblement l'évolution du prototype, dans une phase où le programmeur avance surtout par tâtonnement.
- Si les générateurs d'interface facilitent effectivement l'agencement des éléments de dialogue, nous estimons que le programme généré n'est pas toujours facile à maintenir. La connection avec le noyau de l'application s'effectue dans une zone grise, où les compétences entre le programmeur et le générateur se chevauchent. Il devient difficile d'entreprendre des retouches lorsque ces liens ont été tissés.

Pour programmer l'interface graphique, nous avons choisi l'environnement de programmation Tcl-Tk [165], pour les raisons suivantes :

- Tcl-Tk s'impose comme un standard, notamment dans le monde Unix;
- ce langage de script, c.-à-d. interprété, permet de créer très rapidement une ébauche de prototype;
- le langage est largement appuyé par la communauté scientifique, qui met à disposition quantité de ressources [92] (documentation, paquetages graphiques, extensions du langage, générateurs d'interface, debuggers, etc.).

Afin de structurer au mieux les sources de l'interface utilisateur, nous utilisons l'extension orientée objet la plus répandue, à savoir itcl-itk [136].

6.3.2 Interface graphique basée sur Tcl-Tk

Au niveau conceptuel, nous avons attribué la responsabilité de l'interface graphique à un ensemble d'*agents de dialogue*. Nous allons maintenant concrétiser l'architecture, et aborder la réalisation d'une interface en Tcl-Tk. Nous mettons l'accent sur trois aspects de l'implémentation : la simulation d'agents en Tcl, l'organisation des sources selon une méthodologie orientée objet, et le pilotage des éléments de dialogue.

Implémentation des agents de dialogue

Au niveau conceptuel, l'interface homme-machine est modélisée comme une société d'agents de dialogue. En guise d'implémentation, nous représentons chaque agent par un objet itcl [136]. Cette mise en oeuvre est idéale en terme d'*encapsulation*, car le concept d'objet est une concrétisation très proche de la philosophie multi-agents.

Quant au mécanisme d'*activation*, le programmeur a le choix entre plusieurs variantes, présentées dans le tableau 6.1 (cf. aussi annexe A) :

Expression	Mécanisme d'activation
`thread create "$destinee doSomething"`	`Création d'un thread (PtTcl)`
`after 0 "$destinee doSomething"`	`Postage d'événement (Tk)`
`$destinee doSomething`	`Appel de procédure (Tcl)`

Tableau 6.1 : Simulation de l'activation d'un agent en Tcl.

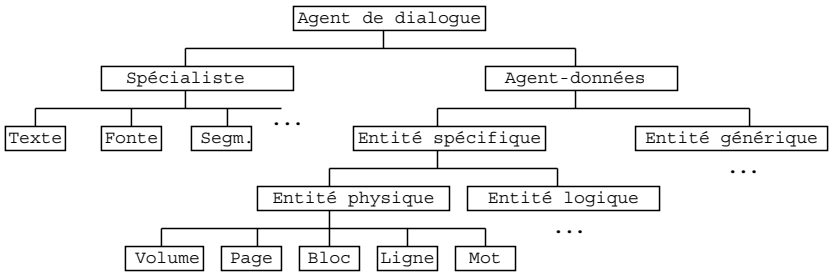


Figure 6.9 : Hiérarchie de classes pour les agents de dialogues

- La réalisation la plus proche du modèle conceptuel consiste à créer de véritables threads préemptif, à l'aide de l'extension PtTcl [81]. Dans ce cas, l'exécution des agents sera vraiment concurrente.
- La réalisation la plus éloignée revient à exécuter des appels directs de procédures. Il s'agit d'une forme dégénérée d'interaction, puisque l'expéditeur perd le contrôle jusqu'à ce que le destinataire ait complétement réagi.
- Une solution intermédiaire, qui préserve l'asynchronisme mais pas la concurrence, consiste à poster l'exécution de la procédure en tant qu'événement. Avec la commande `after` de Tk, le programmeur ajoute dans la boucle d'événement un script à exécuter plus tard. Avec cette méthode, l'agent n'est pas interrompu par l'envoi d'un message.

Dans notre application, où les agents de dialogue n'ont pas a priori à exécuter des traitements coûteux, même l'application directe des méthodes suffit à donner l'illusion d'un comportement concurrent. Nous avons donc choisi l'application directe de méthodes pour mettre en oeuvre les interactions entre agents de dialogue. Les interactions en provenance des agents d'analyse sont en revanche traitées par l'entremise du gestionnaire d'événements (cf. section 6.3.3).

Conception orientée objet

Au niveau des techniques de programmation, l'approche objet simplifie la définition des différentes sortes d'agents de dialogue, grâce au mécanisme d'héritage. La figure 6.9 présente une hiérarchie naturelle entre les classes d'agents. La classe racine offrira les fonctionnalités de communication avec le noyau d'analyse. On peut aussi y définir des moyens de traçage, pour faciliter la phase de développement. Cette classe de base se spécialise en agents-données et en spécialistes.

Tous les spécialistes ont en commun la gestion des travailleurs alloués et les analyses en cours. Ils diffèrent par contre sur les panneaux de configuration propres à chaque domaine d'analyse.

Quant aux agents-données, qui offrent tous des moyens de visualisation et d'édition, ils se distinguent selon qu'ils correspondent à des entités spécifiques ou génériques. Dans le premier cas, les sortes d'agents maintiennent une référence directe vers les images de documents. Dans le deuxième cas, les agents forment une partie du modèle de document.

Widgets

La tâche principale des agents de dialogue est de piloter des éléments de dialogue (widgets). Les widgets sont les constituants de l'interface graphique, alors que les agents de dialogue servent à structurer les relations avec le noyau de l'application.

Notre plateforme favorise la structuration des éléments de dialogue. En effet, l'extension itcl-itk [136] offre la possibilité de définir, à haut niveau, de nouvelles classes de widgets, par spécialisation ou agglomération de widgets existants.

Pour notre application, voici quelques éléments de dialogue souhaitables :

- une fenêtre de visualisation des pages de documents, où les différents éléments reconnus se superposent aux images;
- des lentilles magiques (cf. annexe C) pour gérer la superposition des informations, notamment dans la fenêtre principale;
- un visualisateur de structure hiérarchique, pour naviguer plus aisément dans les structures de documents;
- des menus contextuels attachés à chaque manifestation visuelle des agents de dialogue, par exemple le rectangle détourant une ligne;
- des panneaux de configurations pour gérer la paramétrisation des spécialistes.

Les liens entre widgets et agents de dialogue sont bidirectionnels. Les agents de dialogue créent les widgets, ou les reconfigurent en fonction de l'évolution de la solution courante. Les widgets se chargent d'intercepter les événements qu'ils subissent, et les redirigent vers les agents de dialogue concernés. Un widget peut être dédié à un agent de dialogue, mais rien n'empêche d'en partager la gestion entre plusieurs agents. L'important, c'est de mettre en oeuvre un mécanisme qui permet à l'utilisateur de désigner, à travers ses interventions, les unités qui forment le coeur du système.

Même avec ce cadre d'implémentation, le développeur devra encore préciser certains choix sur le *partage des compétences* entre la partie interface graphique et le noyau d'analyse. Supposons par exemple qu'on désire offrir une fonctionnalité pour effacer le texte d'un bloc. Plusieurs solutions sont imaginables. On pourrait ajouter un article dans le menu attaché au bloc, puis rediriger cette commande vers l'agent d'analyse du bloc. Dans ce cas, on laisse entendre que c'est une opération suffisamment générale pour en déléguer la traduction au moteur d'analyse. Une autre approche consisterait à sélectionner tous les mots du bloc, et à leur appliquer une même opération d'effacement. Avec cette deuxième réalisation, c'est l'interface graphique qui se charge de décoder la commande, et ce sont les agents mots qui avertissent leur équivalent dans l'espace d'analyse. Ce n'est plus l'opération «supprimer le texte d'un bloc» qui a un sens dans le moteur d'analyse, mais l'opération «mettre à jour le texte d'un mot». Le choix dépendra notamment des autres fonctionnalités, en vue d'équilibrer les compétences des deux populations d'agents.

6.3.3 Couplage expressif

A notre avis, la clé de réussite d'une réalisation multi-langages réside dans l'expressivité du couplage entre les langages de programmation[1]. Les interactions entre modules hétérogènes devraient se formuler à un haut niveau, c'est-à-dire sans encombrer le programme source avec des détails de mise en oeuvre. C'est un aspect que nous avons particulièrement approfondi, ce qui a aboutit à une approche généralisée au couplage de n'importe quel environnement de programmation (cf. annexe A).

Dans le cas de notre projet *CIDRE*, le couplage doit permettre d'exprimer des interactions entre des threads PT-PVM et des objets itcl, comme prévu dans la conception multi-agents. Au niveau du contrôle, tout le code de l'interface utilisateur, avec ses agents de dialogue, s'exécute dans un unique processus lourd Unix, à travers un interpréteur Tcl-Tk (un programme `wish`). Les canaux inter-processus sont des ressources limitées sous Unix, et cette contrainte nous incite pratiquement à canaliser dans un seul flux toutes les communication entre l'espace de calcul PT-PVM et les objets itcl, puis à rediriger les messages au bon correspondant. La situation se présente donc comme sur la figure 6.10 : les sources du programme expriment les communications point-à-point, alors que physiquement, les informations transitent par un pôle de chaque côté de la frontière des langages. Avec notre approche, le canal de communication est certes un goulet d'étranglement. Toutefois, il ne fait que refléter le débit limité de la connection physique (pipeline Unix). Le *traitement* des messages est lui bel et bien décentralisé; ce n'est que l'*acheminement* qui passe par un canal central. Par ailleurs, on peut réduire la quantité d'information à transmettre en recourant au système de fichiers. Par exemple, il sera certainement plus agréable de communiquer une image sous forme de fichier TIFF; dans ce cas, seul le nom du fichier transite réellement sur le canal.

La figure 6.11 illustre l'expressivité du codage. Un seul appel (a) suffit à l'agent d'analyse pour créer à distance son agent de dialogue, et établir le lien de communication de manière transparente. Par la suite, l'envoi de messages vers itcl (b) activera automatiquement ce correspondant privilégié, sous forme d'invocation de méthode. De son côté, l'agent de dialogue utilise le même niveau d'abstraction (c) pour envoyer les informations vers l'agent d'analyse. Les types de messages sont désignés par des identificateurs analogues dans les deux parties. Dans notre implémentation, nous avons profité du fait que Tcl manipule exclusivement des chaînes de caractères, ce qui permet immédiatement d'exporter l'accès à n'importe quelle construction.

[1] et une des clés de réussite d'un programme interactif tient à l'expressivité du couplage entre l'interface et le noyau.

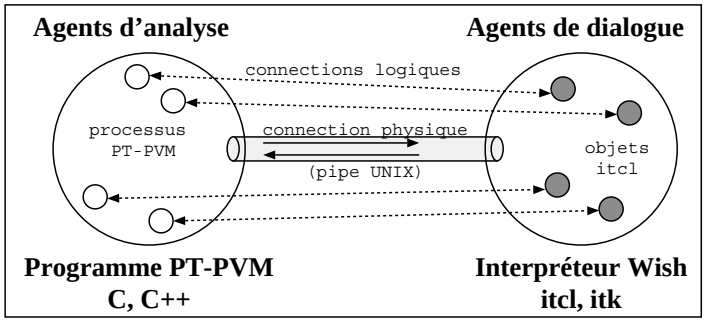


Figure 6.10 : Frontière inter-langages dans le système multi-agents.

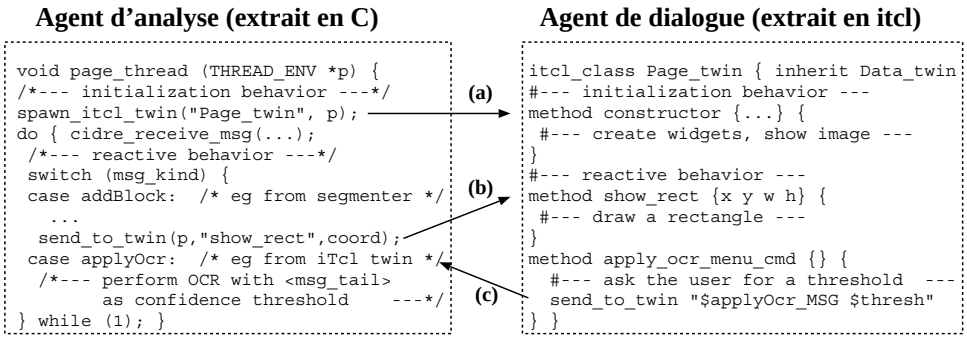


Figure 6.11 : Expressivité des interactions entre agents d'analyse et de dialogue.

6.4 Aperçu du prototype réalisé

Conformément au cahier des charges, notre plateforme logicielle est suffisamment générale pour servir de base à toutes sortes d'applications. Afin de tester l'applicabilité de l'architecture, nous avons tout de même développé un petit prototype. Le programme que nous présentons ici est limité au sous-problème de la reconstitution de la *structure physique* de documents quelconques. La section 6.4.1 énumère les caractéristiques du prototype. L'intérêt principal de cette description tient à la synthèse des décisions laissées ouvertes dans l'architecture : toute application basée sur *CIDRE* pourra être résumée de manière analogue dans ce genre de *fiche technique*. La section 6.4.2 illustre le fonctionnement de l'environnement interactif à travers quelques étapes d'un scénario d'exécution.

6.4.1 Caractéristiques techniques de notre prototype

Application Le prototype est restreint au sous-problème de la reconnaissance de la structure physique complète. A partir d'images de pages, il s'agit de reconstituer la découpe en blocs, lignes et mots, et de déterminer correctement le texte et les fontes véhiculées. La notion de modèle de documents est volontairement absente. Le prototype n'est pas consacré à une catégorie de documents particulière, mais les traitements portent sur les parties textuelles. La solution finale est fournie sous forme de fichier DAFS.

Agents-données Les agents-données se résument à des *entités physiques spécifiques* de cinq types : volume, page, bloc, ligne et mot. Ils s'exécutent dans un unique processus lourd, en compagnie des spécialistes; par ailleurs, le niveau mot est simulé par les agents lignes, comme décrit dans la section 6.2.2.

Spécialistes et travailleurs Il y a trois spécialistes, consacrés respectivement au texte, à la fonte, et à la segmentation. Ils ont un accès direct à l'arbre DAFS. Chacun pilote son ensemble de travailleurs. Il n'y a qu'une sorte de travailleur par spécialiste. Pour la reconnaissance de fontes, le travailleur est écrit en C++, illustrant ainsi le mélange de programmes C et C++ dans la plateforme PT-PVM. L'exploitation du parallélisme est discutée plus bas.

Analyseurs automatiques Plusieurs outils d'analyse ont été intégrés au prototype. Pour la segmentation, nous nous basons sur le paquetage complet issu de la thèse d'Azokly [10], complété par deux outils simples basés respectivement sur le profil de projection et l'algorithme RLSA. La méthode courante est choisie

par l'utilisateur. L'OCR principal correspond au logiciel commercial ScanWorX [29], et nous sommes sur le point d'intégrer notre OCR monofonte. Pour la reconnaissance de fontes, nous avons tiré profit du système *ApOFIS*, développé par Zramdini [213].

Style de dialogue homme-machine Le système est entièrement sous contrôle de l'opérateur. Les analyses automatiques sont explicitement commandées par l'utilisateur (cf. section 3.4), qui désigne le type de traitement et les entités concernées. A mesure que les résultats sont trouvés, ils viennent s'afficher dans l'interface de manière asynchrone. Dans son état actuel, le prototype ne prend donc pas d'initiative, mais on peut facilement aboutir à une forme triviale de collaboration faiblement couplée (cf. section 3.4.3). Par exemple, la modification suivante a été testée. Une commande de segmentation (resp. d'OCR et de reconnaissance de fontes) est générée lors de constuction d'un objet itcl du type page (resp. de type bloc et ligne). Avec cette modification (trois lignes de code Tcl), le prototype manifeste une certaine autonomie, puisque les traitements sont lancés automatiquement, sans que l'utilisateur s'en aperçoive ou soit gêné dans son travail. Pour le moment, nous n'avons pas mis en oeuvre de dialogue homme-machine plus évolué.

Aspects ergonomiques La fenêtre principale présente la page courante, sur laquelle viennent se superposer les résultats trouvés. A chaque élément graphique correspond un menu contextuel activé par le bouton droit. Les pages ont à l'écran les mêmes dimensions que la version papier. Nous avons mis en oeuvre des lentilles magiques (cf. section 3.1.3 et annexe C), afin de pouvoir visualiser séparément les différentes couches d'information, ainsi que l'image originale agrandie.

Adaptabilité Les fonctions d'apprentissage incrémental portent sur l'OCR et la reconnaissance de fontes, et sont déclenchées par des commandes explicites (cf. section 3.2.2), au même titre que les commandes d'analyse. Pour l'OCR, nous utilisons les fichiers de connaissances de ScanWorX; tout est prêt pour intégrer notre outil de post-correction de texte (cf. section 8.6). Pour la reconnaissance de fontes, nous mettons à jour les modèles de fontes d'*ApOFIS*.

Exécution distribuée Le nom des stations de travail incorporées dans la machine virtuelle est fourni par une variable d'environnement. Chaque spécialiste place un travailleur sur chaque processeur. Pour chaque requête par la suite, le travailleur est choisi selon une politique de tourniquet (round-robin). Les informations sont transmises par des fichiers intermédiaires. Le processus exécutant l'interface graphique en Tcl peut être placé sur n'importe quelle machine du réseau.

6.4.2 Exemple de scénario

Afin d'illustrer le fonctionnement de notre prototype dans sa version actuelle, nous allons discuter quelques étapes typiques survenant durant la reconnaissance d'un échantillon.

Le scénario que nous décrivons débute dans la situation suivante. Après avoir lancé le programme, l'utilisateur vient de charger une première image de page. Deux agents-données évoluent dans le système, l'un correspondant au volume, l'autre à la nouvelle page. Lors de sa création par l'agent volume, l'agent d'analyse page crée son équivalent dans l'espace de dialogue, qui affiche immédiatement l'image réduite dans la fenêtre principale. Les trois spécialistes ont été engendrés au démarrage du programme. Leur tâche initiale est de générer l'ensemble des travailleurs, ainsi qu'un agent de dialogue.

Comme illustré sur la figure 6.12, l'opérateur décide maintenant d'appliquer une segmentation sur cette image. Pour ce faire, il invoque le menu contextuel de l'agent page, et sélectionne l'article désiré. Cet événement a pour effet d'activer l'agent de dialogue via une invocation de méthode itcl. En décodant le message, l'agent de dialogue décide de rediriger cette requête à l'agent d'analyse; l'interaction se transforme en messagerie PT-PVM. A son tour, l'agent d'analyse réagit au message en envoyant une requête d'analyse au spécialiste de segmentation. Le spécialiste prend connaissance des informations sur la page (notamment un accès à l'image), choisit l'outil de segmentation et ses paramètres, estampille la requête avec l'identificateur de l'agent-donnée, puis délègue à un travailleur le traitement proprement dit.

Durant la segmentation, le système n'est pas bloqué. L'utilisateur pourrait par exemple commencer à segmenter à la main les parties où il s'attend à des imprécisions de l'analyseur automatique; les agents sont prêts à réagir en conséquence. Quand l'algorithme d'analyse s'achève, le travailleur sauvegarde les résultats dans un fichier temporaire, et en avertit le spécialiste. Celui-ci lit la solution DAFS proposée, et la retourne à l'agent donnée. L'agent page essaie de fusionner cette nouvelle interprétation avec la découpe qu'il avait jusqu'alors. Dans notre cas, la nouvelle interprétation est compatible, et l'agent page crée alors un agent pour chaque bloc proposé par la segmentation.

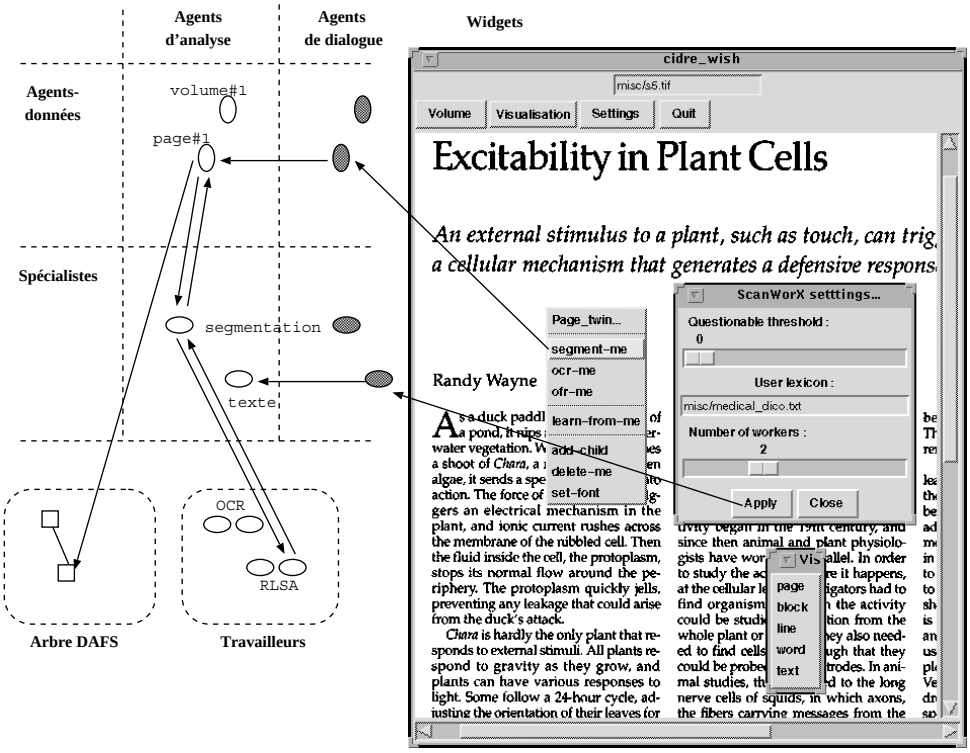


Figure 6.12 : Scénario – Requête de segmentation.

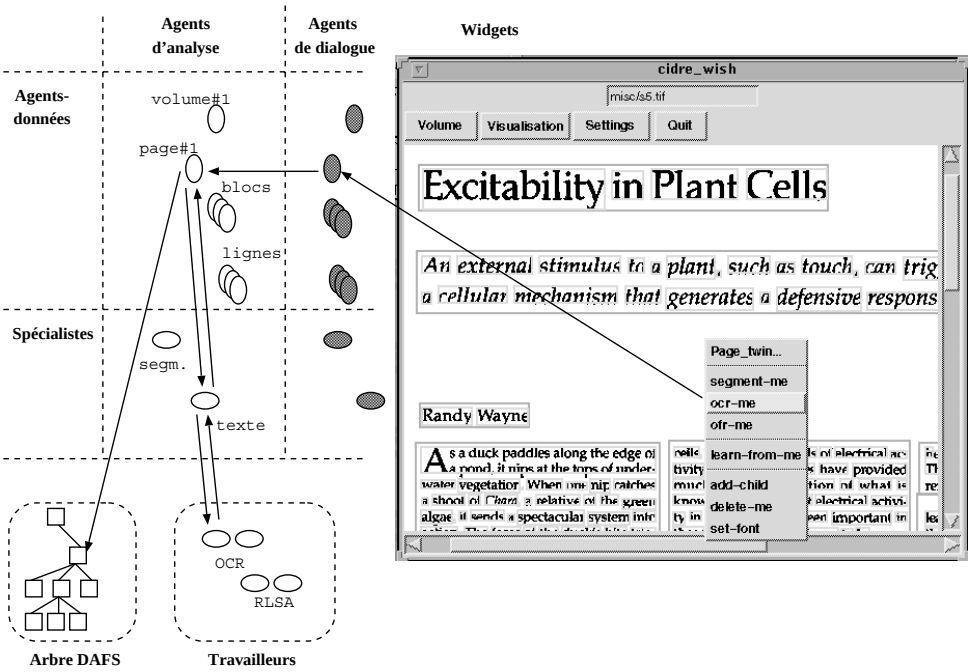


Figure 6.13 : Scénario – Requête d'OCR.

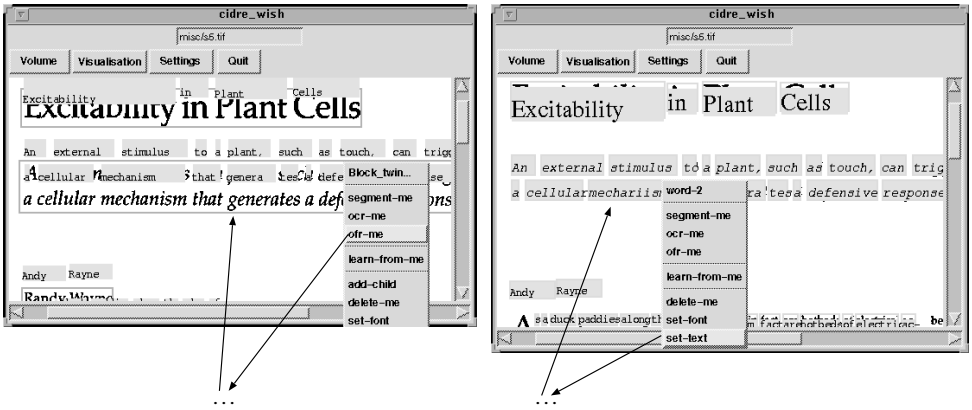


Figure 6.14 : Scénario – Reconnaissance de fonte et édition manuelle.

Le comportement initial d'un bloc provoque la création d'un agent de dialogue, qui va dessiner un rectangle pour délimiter la surface de l'image. En outre, le bloc consulte le noeud DAFS qu'il doit gérer, afin de générer le cas échéant les agents ligne qui lui seraient déjà affectés. On constate sur la figure 6.13 qu'à la fois la population d'agents et l'affichage graphique se sont enrichis. A ce moment, l'utilisateur commande l'OCR sur toute la page, ce qui sera accompli par un recours à ScanWorX. Le déroulement est alors analogue à la segmentation précédente, si ce n'est que lors de la réception des résultats, les agents-données devront fusionner les informations avec les hypothèses de segmentation. Un conflit a surgi sur le premier paragraphe, qui était mal segmenté à cause d'une lettrine. Pour l'interprétation en lignes, ce prototype accorde a priori plus de confiance à l'OCR qu'à la segmentation, et c'est donc la bonne solution qui est conservée.

A mesure que les différents mots sont acceptés par les agents ligne, l'utilisateur voit apparaître le texte à l'écran. Il peut alors corriger les erreurs de reconnaissance, ou préférer se concentrer sur les changements de fontes. Sur la figure 6.14, la reconnaissance de fonte est appelée sur le deuxième bloc. Par la suite, l'utilisateur corrige manuellement une erreur d'OCR.

Conclusion

La plateforme que nous avons choisie pour réaliser l'architecture logicielle de *CIDRE* repose sur la programmation concurrente et distribuée. Pour programmer l'interface utilisateur, nous proposons l'environnement Tcl-Tk, que nous avons couplé de manière élégante au moteur d'analyse. Nous avons développé un premier prototype qui a permis de valider notre architecture, en intégrant dans un environnement interactif la segmentation, l'OCR, et la reconnaissance de fontes. Globalement, l'architecture logicielle et le prototype se caractérisent par une grande souplesse.

Le chapitre suivant aborde une autre question importante : comment concevoir des analyseurs qui s'intègrent bien dans notre architecture assistée.

Chapitre 7

Méthodes d'analyse

Notre architecture n'apporte jusqu'ici aucune contribution aux algorithmes de reconnaissance. Ce chapitre et le suivant comblent en partie cette lacune, en étudiant le développement d'outils d'analyse pertinents pour le projet *CIDRE*.

L'abandon d'une reconnaissance entièrement automatique au profit d'un environnement assisté conduit à remettre en question les techniques d'analyse et leur mise en oeuvre. D'une part, les analyseurs devront réagir aux interventions de l'utilisateur, notamment en mettant à jour leurs connaissances par apprentissage. D'autre part, la souplesse de l'architecture *CIDRE* multiplie les contextes d'utilisation possibles : durant une session de reconnaissance, un analyseur peut être sollicité dans différentes conditions, et il s'agit de faire face à une variété de scénarios.

Ce chapitre dessine une nouvelle tendance dans la conception d'analyseurs automatiques capables de s'intégrer au mieux dans notre architecture interactive. La première section amène une redistribution des priorités dans les critères qui prévalent lors du développement d'un analyseur. Dans la deuxième section, nous passons en revue les divers types de relations que les analyseurs peuvent entretenir entre eux, et montrons comment exploiter ces interdépendances de manière collaborative. Enfin, la section 7.3 présente une démarche qui met le savoir-faire typographique au service des méthodes d'analyse d'images de texte, avec des conséquences positives pour l'interface graphique comme pour l'exploitation des résultats de reconnaissance.

7.1 Critères de qualité

La conception et la réalisation d'analyseurs automatiques pour la reconnaissance de documents sont conditionnées par les priorités établies dans le cahier des charges. Habituellement, les choix techniques sont essentiellement dictés par des critères de performances, à la fois en temps de calcul et en précision des résultats. La philosophie du projet *CIDRE* accorde une vive importance à d'autres critères de qualité, en vue de favoriser (i) le pilotage au sein d'un environnement assisté, (ii) l'amélioration des performances à l'usage, et (iii) la réutilisabilité dans des applications variées. Cette section passe en revue quatre principes directeurs qui devraient mieux être pris en compte.

7.1.1 Simplicité

A la base de la plupart des algorithmes d'analyse, il y a une idée simple. Mais à mesure qu'ils progressent dans la réalisation ou les tests, les développeurs sont incités à retoucher certains détails de la méthode originelle. Durant l'expérimentation, il est en effet frustrant de constater un gaspillage manifeste de temps de calcul, ou de flagrantes erreurs de reconnaissance. C'est donc dans un esprit louable que l'algorithme est enrichi grâce à des heuristiques, qui améliorent d'ailleurs souvent le comportement de manière très pertinente. Pourtant, nous prétendons que cette attitude est dangereuse à plusieurs égards :

- En phase de test, le développeur manque de recul vis-à-vis du problème, car son attention se fixe sur quelques échantillons, pour lesquels il est pressé d'obtenir des résultats conformes aux espérances. Dans ces conditions, on risque de trouver dans les sources certains seuils fixés empiriquement, ou des traitements exotiques consacrés individuellement à quelques cas récalcitrants, mais sans fondement théorique.
- Des modifications hasardeuses constituent une entrave à la consistance de la méthode, et compliqueront les extensions conceptuelles. Par exemple, il devient très ardu d'ajouter une fonctionnalité d'apprentissage incrémental à une méthode de reconnaissance encombrée de traitements exceptionnels.
- En l'occurrence, et comme souvent en informatique, il est utopique d'anticiper tous les usages futurs d'un programme. Ce qui paraît négligeable un jour peut devenir crucial le lendemain. Dès lors, toutes les hypothèses méritent d'être clairement spécifiées, au lieu d'être dissimulées au milieu du code.

C'est pourquoi nous conseillons d'utiliser des techniques de reconnaissance simples, et de concentrer les efforts sur l'intégration cohérente des analyseurs dans la globalité du système. A terme, cette approche sera plus bénéfique qu'une sur-optimisation de détails algorithmiques masqués par une interface hermétique. Il n'y a plus beaucoup d'opportunité d'améliorer les analyseurs existants pris individuellement. Les progrès sont dorénavant attendus à travers les interactions entre analyseurs, et la capacité d'adaptation en cours d'exécution.

7.1.2 Paramétrisation

Concilier la clarté d'un algorithme d'analyse avec des performances moyennes acceptables implique souvent une paramétrisation fine. Un analyseur gagne évidemment en généralité s'il peut être reconfiguré avec souplesse, sans retouche des sources. Le risque est alors de déplacer les difficultés vers l'extérieur du code, au moment du choix des paramètres. En fait, il ne suffit pas de sortir du programme la valeur d'une constante pour en faire un bon paramètre. La configuration devrait être suffisamment intuitive pour que l'utilisateur puisse la comprendre, voire anticiper son influence sur le comportement de l'analyseur, que ce soit en rapidité ou en précision. Le tableau 7.1 évoque certaines options de configuration typiques pour les analyseurs de documents. Naturellement, la spécification exacte de l'ensemble des paramètres va dépendre de la technique de reconnaissance utilisée.

D'une manière générale, nous considérons que la gestion des paramètres est aussi importante que l'algorithme d'analyse. Pour faciliter l'utilisation d'un analyseur, plusieurs solutions s'offrent aux développeurs. Tout d'abord, il s'agit de produire une documentation aussi claire que possible, qui donne une sémantique à chaque paramètre. Le paquetage peut aussi offrir un paramétrage par défaut, ou des jeux de paramètres prédéfinis pour différentes situations typiques. Il est parfois fort utile de prévoir une interface graphique pour visualiser et choisir les paramètres. On peut encore réaliser de vrais programmes d'estimation automatique de paramètres, qui proposent un paramétrage adéquat en fonction d'un échantillon de données typique, voire du résultat attendu; ce dernier cas se rapproche conceptuellement de l'apprentissage (cf. section 7.1.3).

Type d'analyseur	Paramétrage
Reconnaissance de fontes	Ensemble de fontes candidates, critères d'homogénéité
Segmentation	Espacement minimal horizontal et vertical, contraintes sur l'homogénéité des dimensions ou des espacements, taille des fontes
OCR	Alphabet, fonte, probabilités a priori des caractères, langue, lexique, seuil de rejet
Etiquetage logique	Modèle de documents, tolérance à la syntaxe, tolérance à la présentation

Tableau 7.1 : Quelques types de paramètres possibles.

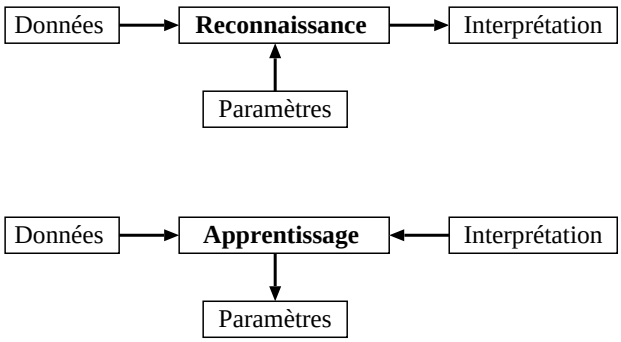


Figure 7.1 : Dualité entre reconnaissance et apprentissage.

7.1.3 Apprentissage incrémental

L'aspect le plus décourageant d'un analyseur automatique n'est pas tellement la valeur de son taux d'erreurs, mais bien plutôt la répétition de fautes systématiques. A ce propos, l'attitude vis-à-vis des performances se révèle déterminante. Si on se fixe comme objectif irréductible une reconnaissance 100% automatique, alors les erreurs sont intolérables. Dans cette optique, offrir une forme d'apprentissage, c'est admettre que la méthode est perfectible. Pour notre part, nous trouvons que toute technique d'analyse est par nature imparfaite, et que les erreurs sont un phénomène inhérent au problème. Il faut dont prévoir des moyens (interactivité, apprentissage) qui réduiront les effets de cette imprécision. L'adaptabilité nous paraît une qualité indispensable du système de reconnaissance. Il ne faut pas espérer prévoir toutes les situations en constituant auparavant un ensemble d'échantillons destinés à l'entraînement du système. Le but est finalement d'optimiser le rendement de l'analyseur *dans les conditions d'exploitation*, et non lors des tests qui suivent la phase de développement.

La figure 7.1 schématise la dualité entre les processus de reconnaissance et d'apprentissage. Le terme *paramètres* désigne ici, au sens large, la base de connaissances utilisée par la méthode d'analyse. Ces informations constituent la partie variable de l'expertise offerte, par opposition à l'algorithme lui-même qui est immuable[1]. En pratique, la paramétrisation n'implique aucune recompilation. La base de données peut contenir diverses structures de données : description d'exemples positifs et négatifs, vecteurs de caractéristiques, seuils, règles de syntaxe, protocoles de pré- ou post-traitement, options, poids dans un réseau de neurones, etc.

La plupart des composants d'un système de reconnaissance de documents mériteraient de fournir une forme incrémentale d'apprentissage. Voici quelques exemples concrets, avec lesquels nous avons acquis une certaine expérience :

- *Seuils de segmentation* – La majorité des techniques de segmentation (RLSA, profil de projection, fusion de composantes connexes, etc.) utilisent des seuils, typiquement pour l'espacement minimal entre composants. Azokly [10] montre comment ces seuils peuvent être estimés automatiquement à partir d'une image typique. Sa méthode a été motivée par une analyse statistique de la distribution des espacements entre composantes connexes. Un véritable apprentissage est également possible, en analysant les espacements dans une portion d'image correctement segmentée. S'il est parfois impossible de trouver un paramétrage tel que le résultat soit entièrement correct, on peut en revanche trouver les seuils qui minimisent les erreurs.

[1] Cette distinction n'est qu'informelle puisqu'à l'extrême, rien n'empêche l'algorithme de simuler une machine universelle, et les paramètres de représenter un programme !

- *Reconnaissance de fontes* – Le système de reconnaissance de fontes a priori *ApOFIS* de Zramdini [215, 213] se base sur des caractéristiques globales extraites de l'image d'une portion de texte. Un modèle de fonte est représenté par une fonction de distribution paramétrique, comprenant la moyenne et la variance de chaque caractéristique. Cette technique d'analyse offre immédiatement l'apprentissage incrémental, qui plus est sous la forme la plus intuitive possible : l'utilisateur n'a qu'à désigner une portion de texte et le nom de la fonte correspondante, et la base de connaissances s'adapte en conséquence.
- *Etiquetage logique* – Dans une autre partie du projet *CIDRE*, Rolf Brugger [38] montre comment un modèle statistique de structure de documents se révèle adéquat pour (i) véhiculer correctement la syntaxe et les règles de présentation, (ii) diriger un algorithme efficace d'étiquetage logique, et (iii) construire un modèle de manière incrémental, à partir d'exemples étiquetés. Ce travail met en évidence une forme d'apprentissage simple, dans un domaine où l'acquisition de connaissances passait habituellement par des techniques lourdes d'inférence grammaticale.
- *Logiciel d'OCR commercial* – Certains OCR commerciaux offrent la possibilité d'enrichir les connaissances, par exemple pour s'adapter à un type de dégradation, voire à un nouvel alphabet. Nous avons été amenés à explorer le mode d'apprentissage du logiciel ScanWorX [29], en vue d'en rendre le pilotage plus convivial (cf. section 8.5).
- *Post-correction d'OCR* – La section 8.6 évoque un algorithme de post-correction automatique d'OCR. Le principe consiste à mémoriser chaque correction manuelle, pour rechercher ensuite d'autres occurrences de cette erreur dans une phase de post-traitement des résultats d'OCR.

A côté de ses avantages manifestes, l'apprentissage incrémental soulève aussi plusieurs difficultés. Premièrement, les méthodes d'apprentissage doivent affronter les problèmes de *stabilité* ou de *convergence*. Il est souvent difficile de garantir qu'une transaction d'apprentissage ne perturbera pas les connaissances déjà acquises. A l'extrême, l'apprentissage risque d'engendrer de nouvelles erreurs encore plus coûteuses. Deuxièmement, les méthodes d'analyse ne supportent pas toutes une forme d'apprentissage élégante. Enfin, l'intégration d'une fonction d'apprentissage au sein d'un système complet soulève plusieurs questions : quel composant choisit les jeux de paramètres, quand autorise-t-on l'apprentissage, comment défaire une opération d'apprentissage, etc.

A noter encore qu'un analyseur peut essayer de s'améliorer sans intervention extérieure [145]. Supposons qu'on dispose d'un OCR qui est fiable à 95%. Durant la phase d'exploitation, on peut utiliser les caractères reconnus correctement pour ajuster encore plus finement la base de connaissances. Par effet de bord, il est possible que cette adaptation conduise ensuite à mieux distinguer d'autres caractères. Cette approche suppose qu'on est capable de garantir une haute fiabilité sur une partie des résultats, p. ex. en augmentant le taux de rejet [46].

7.1.4 Compatibilité

En reconnaissance de documents, un outil d'analyse automatique est un composant logiciel qui véhicule une certaine expertise, susceptible d'être réutilisée dans différentes applications. D'un point de vue économique, on parlerait de produit à haute valeur ajoutée. Il convient donc de fournir l'effort nécessaire pour assurer une certaine compatibilité. Plusieurs aspects sont concernés :

- séparation claire des fonctionnalités (segmentation, OCR, etc.);
- standardisation des formats de données (p. ex. avec DAFS);
- conventions sur le calcul des valeurs de confiance;
- fonctionnalités de révision ou d'apprentissage;
- portabilité des sources.

Nous estimons que le critère de compatibilité demeure sous-estimé. Les développeurs qui proposent des outils d'analyse ne sentent pas le besoin de se conformer aux standards, et les environnements complets sont souvent câblés aux analyseurs automatiques. Premièrement, les tentatives de standardisation sont relativement récentes. Deuxièmement, les chercheurs sont plus motivés par l'intérêt scientifique d'une nouvelle méthode que par les aspects de mise en oeuvre. Troisièmement, les industriels craignent peut-être une concurrence trop directe si les différents produits sont fonctionnellement équivalents. Pourtant, la discipline de reconnaissance de documents aurait tout à gagner si les analyseurs existants devenaient interchangeables.

7.2 Interdépendances entre analyseurs

Un système complet de réingénierie de documents incorpore plusieurs analyseurs automatiques, et les interdépendances sont inévitables. Or, il y a un décalage notoire entre le test d'un outil pris isolément et son pilotage au sein d'un logiciel intégré. Sur ce point, *CIDRE* critique les architectures monolithiques, et insiste sur une conception soignée des sous-systèmes et de leurs relations. La réutilisabilité est ainsi encouragée, que ce soit pour bâtir une application avec des outils déjà disponibles, ou pour adapter un prototype à une nouvelle application cible. Cette section vise à résumer les principales sortes de combinaisons entre des analyseurs typiques. C'est en effet en exhibant des solutions partielles éprouvées que l'on facilitera la tâche du concepteur, un peu comme l'étude des *design patterns* [72] est une aide à la conception orientée objets.

La difficulté majeure dans la gestion de ces interdépendances tient à l'hétérogénéité des outils d'analyse. Spécifier les formats de données ne suffit pas forcément. Supposons typiquement que l'on désire lancer deux analyseurs en compétition, pour choisir le résultat d'après la mesure de confiance retournée. Or, les deux techniques de reconnaissance peuvent produire des valeurs de confiance difficilement comparables, même après normalisation entre 0 et 1. Par exemple, les techniques basées sur les chaînes de Markov mesurent une forme de probabilité, mais qui est le plus souvent infime. Comment comparer cette valeur avec une mesure de confiance qui suit une autre distribution statistique? Même la sémantique d'une valeur de confiance n'est pas univoque, puisqu'on peut vouloir quantifier soit l'adéquation du résultat avec la donnée, soit l'absence d'alternative plausible[2].

7.2.1 Compétition

Une première forme d'interdépendance se présente lorsque plusieurs analyseurs fournissent séparément le même type de résultat. Dans cette situation de compétition, le système de reconnaissance devra choisir à la fois les analyseurs à invoquer et le résultat final à conserver. Voici quelques stratégies envisageables, qu'on peut d'ailleurs combiner :

- *post-analyse :* lancer systématiquement tous les analyseurs, puis combiner les résultats proposés pour déterminer le plus probable [84];
- *analyse intermédiaire :* classer l'ensemble des outils, p. ex. selon leur rapidité, puis lancer successivement les analyses, jusqu'à satisfaire un certain critère, p. ex. obtenir une confiance suffisante;
- *pré-analyse :* inventer une mesure sur les données d'entrée, p. ex. de la qualité de l'image, qui permette de déterminer l'analyseur le plus approprié.

7.2.2 Producteurs-consommateurs

Les relations de producteurs-consommateurs sont si présentes que le fonctionnement des systèmes de reconnaissance est souvent schématisé par un simple pipeline, comme nous l'avions manifesté sur la figure 2.1 de la section 2.2.2. Toutefois, cette représentation occulte le possible recours à diverses options stratégiques :

- *rétroaction :* les consommateurs donnent en retour aux producteurs des indications sur le résultat utilisé, p. ex. lors d'une confiance trop faible à l'étape suivante, susceptible d'être due à une donnée incorrecte;
- *famille de candidats :* chaque étape propage un ensemble de candidats, retardant ainsi le choix de la solution définitive;
- *évaluation paresseuse (lazy evaluation) :* au niveau du contrôle, l'analyse n'est déclenchée que si un autre composant en a déjà demandé le résultat.

7.2.3 Protocoles de coopération

Nous parlons de *protocole de coopération* entre analyseurs lorsque plusieurs analyseurs automatiques *s'influencent mutuellement* dans le schéma d'exécution d'un système de reconnaissance. Les formes d'*influence* typiques comprennent le choix des paramètres d'exécution, ainsi que la modification de la base de connaissances. L'influence peut être directe, comme dans un protocole de négociation explicite [121] entre sources de connaissances, ou indirecte, auquel cas d'autres analyses servent d'intermédiaires.

[2] une manière de prolonger ce point consisterait à expérimenter des analyseurs produisant un couple de valeurs de confiance, p. ex. en adaptant *ApOFIS* ou notre OCR monofonte.

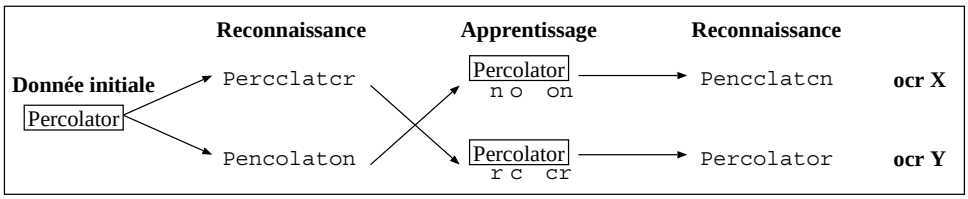


Figure 7.2 : Influence mutuelle par apprentissage croisé.

Par exemple, le protocole schématisé sur la figure 7.2, mélange les fonctionnalités d'apprentissage et de reconnaissance. Supposons que deux OCR basés sur des techniques différentes soient fonctionnellement équivalents et supportent l'apprentissage incrémental. Comme c'est souvent le cas, l'effet d'une transaction d'apprentissage n'est pas spécifié (cf. section 9.2.1), et présente donc une part d'*aléatoire* quant aux futurs résultats de reconnaissance. Il est possible de piloter ces deux analyseurs au-delà d'une simple compétition. Si les résultats initiaux sont différents, on entraîne chaque OCR selon les corrections suggérées par l'autre. Après cette adaptation, les analyseurs donneront peut-être une nouvelle interprétation de la donnée initiale. Dans la figure 7.2, nous laissons entendre que le nouveau résultat produit par l'OCR Y pourrait être correct, *même si la donnée apprise était erronée*. En fait, il y a des risques que le procédé tende plutôt à cumuler les défauts (comme l'OCR X) que les qualités (comme l'OCR Y) des deux outils. Notre but est seulement de montrer que la combinaison d'analyseurs ne se limite pas à additionner les comportements individuels. A travers ces phénomènes, nous rejoignons d'ailleurs le champ d'étude des systèmes multi-agents.

Zramdini [213] donne un autre exemple de coopération entre un OCR monofonte et un outil de reconnaissance de fontes a priori (Optical Font Recognition, OFR). Le principe revient à répéter les appels à l'un puis à l'autre, en modifiant à chaque fois les paramètres en fonction des résultats de l'étape précédente. Dans cette coopération, l'OFR sert à restreindre les fontes candidates possibles. L'OCR sert à partitionner l'entité analysée en deux catégories : les zones où une fonte candidate a pu être validée en raison de la qualité du texte reconnu, et les zones qui semblent formattées avec une autre fonte, et sur lesquelles il faut réitérer le processus. Ce protocole tient compte des constatations suivantes : (i) l'OFR est d'autant plus robuste que la zone est longue; (ii) l'OCR monofonte est une opération coûteuse; (iii) les changements de fontes sont assez rares.

A notre avis, les protocoles de coopération constituent un moyen encore sous-exploité d'améliorer les systèmes de reconnaissance. Afin de susciter l'intérêt de recherches plus poussées, la prochaine section montre qu'en pratique, chaque type de résultat est susceptible d'être produit par des procédés variés. Le cheminement analytique à travers les informations n'est pas unique. Globalement, on doit admettre qu'un système qui impose *une* chaîne de traitements depuis les images jusqu'à la structure logique est forcément réducteur.

7.2.4 Variété des chemins d'analyse

Supposons que le système reçoive une collection d'images (I) et produise une solution formée de quatre catégories de résultats : segmentation (S), fonte (F), texte (T), et étiquette logique (E). Le schéma d'analyse traditionnel en pipeline (cf. figure 2.1) laisse entendre qu'il n'y a pas d'autre alternative (I→S→F→T). Pourtant, suivant les cas, on pourrait produire chaque résultat à partir d'autres données :

- estimer les seuils de segmentation à partir des métriques de la fonte (F→S);
- grouper en bloc des lignes ayant une fonte commune (F→S);
- utiliser un lexique pour détecter les frontières de mots (T→S);
- utiliser le texte et les métriques de fonte pour segmenter en mots (F,T→S) (cf. section 8.4);
- consulter les règles de présentation du modèle de document pour proposer le texte (E→T), la fonte (E→F), voire pour guider la segmentation (E→S) sur une portion étiquetée;
- recourir à un OCR monofonte (F→T);
- valider la fonte par une méthode a posteriori (S,T→F);
- déduire l'étiquette soit de la fonte (F→E), soit du texte (T→E), soit de la segmentation (S→E), lorsqu'il n'y a pas ambiguïté.

On retrouve ici une forme étendue d'un paradoxe célèbre : «il faut segmenter pour reconnaître, et reconnaître pour segmenter». Nous voyons deux approches permettant de tirer profit de cette situation paradoxale, au lieu de la subir.

La première est plutôt procédurale : elle consiste à développer des algorithmes qui combinent au mieux les analyseurs. On peut par exemple mettre en oeuvre un mécanisme de vote [108, 119] ou de révision locale. Cet algorithme devient alors lui-même un nouvel analyseur qui traite un problème plus riche, comme par exemple reconnaître la fonte *et* le texte.

La seconde approche, plutôt déclarative, consiste à réviser le moteur d'exécution, en mettant l'accent sur les données plus que sur le contrôle. Par exemple, une architecture distribuée ou à base de tableau noir ne décrit pas l'enchaînement des traitements, mais les événements qui font déclencher chaque analyse. En somme, le schéma d'exécution sera déterminé en fonction des résultats apparus en cours d'analyse. Avec l'approche assistée de *CIDRE*, ce phénomène prend encore davantage d'importance, puisqu'une source de connaissance universelle, continue, et fiable est rajoutée, en la personne de l'opérateur humain.

Quelle que soit l'approche envisagée, le système de reconnaissance assisté devra piloter les analyseurs en accordant un soin particulier aux protocoles d'échange d'information entre les différentes étapes de l'analyse. Cette situation contraste avec le développement d'outils entièrement automatiques. Dans ce dernier cas, le programmeur retouche le code des analyseurs pour éliminer toutes les surprises quant au traitement du corpus prévu; il peut ensuite se contenter d'enchaîner ces traitements optimisés.

7.3 Exemple d'intégration en typographie

L'un des objectifs du présent travail a été énoncé comme une quête d'intégration de savoir-faire en reconnaissance de documents. Nous allons maintenant montrer dans quelle mesure les algorithmes de reconnaissance eux-mêmes sont amenés à être révisés lorsqu'on prend la peine d'approfondir l'une des disciplines impliquées. En l'occurrence, nous allons nous pencher sur la typographie [19], et faire le lien entre (i) le savoir-faire typographique en production de documents, (ii) les mécanismes actuels de gestion de fontes sur ordinateur, et (iii) l'état de l'art dans le développement de systèmes de reconnaissance.

La section 7.3.1 fait le point sur la manipulation des fontes en reconnaissance de documents, et constate que les systèmes actuels n'impliquent que très superficiellement les aspects typographiques. La section suivante propose l'intégration d'un support logiciel pour gérer les fontes, au sein de notre plateforme de développement. Enfin, la dernière section passe en revue quelques idées d'algorithmes d'analyse qui tirent concrètement parti des connaissances typographiques. Certaines méthodes ont été expérimentées, et sont reprises dans le chapitre 8.

7.3.1 Notion de fonte en reconnaissance de documents

Fonte en typographie

La typographie constitue une discipline-clé en génie documentaire. Une masse de savoir-faire considérable s'est accumulée dans l'art de concevoir des polices de caractères, mais aussi de formatter des documents suivant des principes esthétiques ou scientifiques. Le rôle incontournable de l'ordinateur dans l'édition et la publication modernes a donné naissance à une branche à part entière appelée *typographie numérique* [6, 104, 105, 103], dont l'étude peut nous être fort utile. Un survol de l'état de l'art n'étant pas de mise dans ce travail, le tableau 7.2 se borne à évoquer en vrac quelques éléments manipulés par les typographes. Notons en passant que l'usage des fontes ne répond pas seulement à des critères de présentation : le plus souvent, les attributs typographiques dénotent les intentions de l'auteur du document, par exemple lorsqu'un mot-clé est mis en évidence par du gras.

Fontes et résultats de reconnaissance

A notre avis, l'importance de la typographie en production de documents contraste avec le manque de rigueur dont fait habituellement preuve la communauté de reconnaissance de documents au sujet des fontes. Par exemple, le problème de la reconnaissance de fontes a été largement négligé par rapport à la reconnaissance de caractères. Bien que les logiciels actuels d'OCR [29, 67] sont parfois capables de restituer certains attributs typographiques (gras, italique, chasse fixe, taille), les résultats sont souvent décevants si on les examine en détail. En général, chaque système de reconnaissance encode les résultats typographiques dans des formats maison (cf. le format XDOC de Xerox [29]), sans faire référence aux standards utilisés en gestion des fontes. Sur le plan scientifique, ce n'est que récemment qu'une poignée de techniques de reconnaissance des fontes ont été proposées et validées [215, 189].

alphabet	majuscules, minuscules, ponctuation, chiffres, langue...
polices	chasses fixe et variable, cursif, avec ou sans sérifs...
attributs	graisse, pente, taille, soulignement, ombrage...
composition	ligne de base, ligne supérieure, ligne des 'x'...
anatomie	sérif, hampe, jambage, boucle...
métriques	point d'ancrage, approches, largeur, tables de crénage...
formats	contour, bitmap, TrueType, Type 1, Metafont, F3...
lisibilité	gris typographique, couleur, complexité des formes, anti aliasing...
formattage	interligne, justification, indentation, pied de page, notes marginales...
logiciels	Fontographer, TypeTool, FontLab, ScanFont, FontStudio, Ikarus...
installation	serveur de fontes, chargement, système de cache...

Tableau 7.2 : Eléments de typographie.

Les informations typographiques font partie de la structure physique, et concernent aussi bien le niveau générique que spécifique. Dans les modèles de documents, les indications sur les fontes constituent des contraintes sur la présentation des entités logiques. Dans un échantillon, l'usage des fontes se manifeste sur les images par le dessin particulier du texte[3]. Durant l'étiquetage logique, on procède à une étape de mise en correspondance afin de garantir que les contraintes exprimées dans les entités génériques sont respectées.

Reconnaissance de fontes

En reconnaissance de fontes, Zramdini [213] distingue deux familles d'approches, suivant les informations fournies en entrée. Dans la reconnaissance *a priori*, l'interprétation textuelle n'est pas disponible, et l'analyse doit donc porter uniquement sur l'image. L'approche *a posteriori* fait au contraire la supposition que le contenu des entités textuelles est connu. Les deux optiques ont leurs propres avantages.

Nous proposons une autre distinction entre discrimination et compréhension de fontes, suivant la forme du résultat attendu. La *discrimination* de fontes s'intéresse uniquement à grouper les éléments textuels qui ont une fonte commune, sans chercher à préciser les particularités typographiques. Dans ce cas, nommer les fontes par un numéro suffit à détecter les changements de fontes dans le document. Dans la *compréhension* de fontes, le but est de décrire en détail les polices utilisées, ainsi que leurs attributs (p. ex. Times-Roman-Bold, 12pt). Il s'agit d'un problème plus fort, qui nécessite une spécification stricte des fontes.

7.3.2 Support logiciel pour la gestion des fontes

Pour traiter les fontes de manière rigoureuse, il est primordial de soigner les répercussions, sur l'architecture logicielle, de la notion de police de caractères. L'idée fondamentale que nous défendons revient à intégrer dans la plateforme de développement un support logiciel pour la gestion des fontes.

Rôle d'un support de fontes

Du point de vue du support logiciel, manipuler des fontes implique *deux niveaux de gestion*. D'une part, une fonte représente une structure de données riche qui fournit diverses informations, comme la forme du dessin de chaque caractère imprimable et les métriques. Nous verrons plus loin que ces données sont directement exploitables pour plusieurs sous-tâches de reconnaissance. D'autre part, les fontes sont par nature proches du système d'exploitation (au sens large), puisque la quasi-totalité des applications interactives sont concernées par l'affichage graphique de texte. C'est pourquoi les environnements actuels offrent en standard des moyens pour gérer l'installation des fontes, indépendamment des logiciels qui pourront utiliser ces ressources. Ces fonctionnalités, associées à un certain consensus sur les formats de fichiers, ont pour effet de faciliter l'accès à une immense collection de fontes numérisées, des plus courantes aux plus exotiques.

Considérant les habitudes en reconnaissance de documents, nous préconisons le principe suivant : les systèmes de reconnaissance devraient être conçus pour exploiter au mieux la gestion des fontes offerte par l'environnement logiciel sous-jacent. Plus concrètement, il s'agit d'exprimer le texte reconnu en conformité avec les fontes supportées par le système hôte. En effet, partager une même spécification des fontes contribue déjà à rapprocher les résultats de reconnaissance du cycle de production qu'avait subi le document.

[3] Notons aussi que dans les formats de documents électroniques (PostScript, DVI), les fontes utilisées sont explicitement mentionnées, contrairement à d'autres attributs de mise en page (marges, colonnes, en-tête, etc.). Parfois, ces fontes sont remplacées lors de la restitution selon des tables d'équivalence.

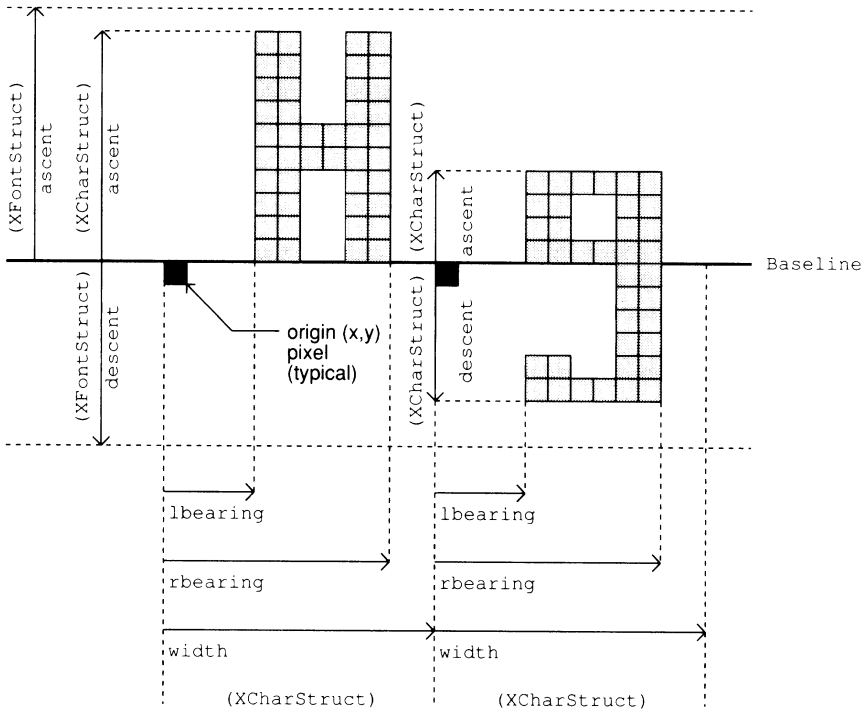


Figure 7.3 : Métriques dans Xlib (extrait du manuel de programmation Xlib).

Plusieurs avantages découlent de cette politique. Ainsi le recours à une spécification stricte des fontes permet de *lever certaines ambiguïtés* sur les formats de sortie, pour la reconnaissance de fontes bien sûr, mais aussi pour la reconnaissance de texte. De fait, le codage ASCII ne suffit pas pour désigner certains caractères (§, ¶, fi, †, ©,γ) de façon univoque. La norme UNICODE étend l'alphabet disponible, mais en toute généralité, seule la référence à la forme dessinée permet de rendre compte de la présence d'un symbole.

Un autre avantage réside dans la possibilité d'*afficher* les résultats de reconnaissance sous leur apparence la plus naturelle, ce qui est particulièrement souhaitable dans un environnement assisté. Par ailleurs, on prépare implicitement le terrain pour les applications dont le but premier est de *réimprimer* des documents dégradés. De même, on facilite la conversion du document reconnu vers des formats d'édition, qui respectent les standards dans le support des fontes. Outre l'OCR pour l'alphabet européen, d'autres tâches de reconnaissance pourraient bénéficier d'une description formelle des objets à identifier. Parmi la multitude de fontes existantes, toutes sortes de notations sont abordées : alphabets non latins (arabe [5], chinois [60]), partitions musicales, formules mathématiques, signes astrologiques ou astronomiques, icônes, dessin, etc.

Support offert par X11

Dans notre plateforme de développement *CIDRE*, nous avons choisi de recourir au support de fontes offert par le système X11 [161]. Cette solution n'est certainement pas idéale en regard de la typographie professionnelle, mais elle offre toutefois des avantages appréciables :

- on a accès aux détails d'une description de fontes, notamment grâce à une spécification précise des métriques (cf. figure 7.3);
- grâce à sa librairie Xlib, X11 peut être piloté depuis l'environnement de programmation C, dans lequel sont écrits nos analyseurs;
- dans sa réalisation Unix, l'environnement Tcl [165] est construit sur X11, et les fontes installées sont immédiatement utilisables dans notre interface graphique;
- X11 prévoit le concept de fontes redimensionnables[4] (*scalable*), ce qui évite d'encombrer inutilement la base de données de fontes par des copies à différentes tailles; en reconnaissance, cela permet en outre de décliner une fonte à différentes résolutions;
- X11 intègre le format de fontes PostScript (Type 1 ou 3), l'un des standards les plus importants;

[4] d'un point de vue typographique, les fontes ne sont *pas* redimensionnables; toutefois, le concept est apprécié quant à la gestion informatique des fontes.

Préfixe	Police	Graisse	Pente	Exemples
	tm-			cidre-tm-r-i-12
cidre-	cr-	r-	r-	cidre-cr-b-r-47
	pl-	b-	i-	cidre-pl-b-i-6
	...			...

Tableau 7.3 : Nommage des fontes dans notre architecture logicielle.

Avec sérifs	Sans sérif	Chasse fixe	Divers
bem Bembo	gis GillSans	cr Courier	zc ZapfChancery
nc NewCentury	hv Helvetica	lst LucidaSansType	zd ZapfDingbats
pl Palatino	ls LucidaSans		sym Symbol
tm Times	ag Avant-Garde		
uto Utopia	hvn HelveticaNarrow		
rkw Rockwell			
cm ComputerModern			
bk Bookman			
lb LucidaBright			

Tableau 7.4 : Quelques polices installées dans notre architecture logicielle.

- X11 est déjà installé dans tous les environnements Unix, ce qui réduit les problèmes de portabilité.

Politique de nommage des fontes

Le nommage des fontes sous X11 n'est pas toujours très élégant : le descripteur de base est une longue chaîne de caractères, plutôt cryptique (X11 Long Font Descriptor, XLFD). Afin d'homogénéiser la référence aux différentes polices, nous avons utilisé pour le nommage la convention résumée dans le tableau 7.3, qui a l'avantage de faire ressortir clairement les attributs typographiques courants (Bold, Italic). La mise en oeuvre de cette politique de nommage profite du concept d'*alias* prévu par X11. Les alias de noms de fonte sont énumérés dans un fichier spécial, et servent à désigner un descripteur XLFD complet par un identificateur, en général raccourci. De surcroît, un alias (p. ex. `mycourier`) vers une fonte redimensionnable peut accepter comme suffixe un nombre entier (p. ex. `mycourier-12`) qui détermine alors la taille nominale en points[5] (on parle alors de «scalable alias»).

Ensemble de fontes

Cette politique de nommage s'est révélée bien adaptée, et compatible avec le modèle utilisé par *ApOFIS*. Le tableau 7.4 énumère les 19 polices qui sont accessibles dans notre architecture logicielle. C'est un ensemble de fontes raisonnable pour conduire nos expériences. Pour une application dédiée, on peut facilement étendre les polices disponibles, par exemple en installant de nouvelles fontes sous X11, et en choisissant un alias selon notre politique de nommage.

Les routines et structures de données offertes par Xlib se sont révélées tout à fait pertinentes pour intégrer les descriptions de fontes et les algorithmes d'analyse.

7.3.3 Apport des connaissances typographiques

Le rapprochement que nous préconisons entre système de reconnaissance et typographie est plus qu'un exercice de style. L'objectif avoué est de profiter des connaissances typographiques pour améliorer les techniques d'analyse. Cette section commence par évoquer les conventions adoptées par les typographes pour rendre un document esthétique. Ce savoir-faire, même s'il est parfois informel, peut servir à émettre des hypothèses lors de la reconnaissance. Le reste de la section se concentre sur un aspect d'architecture logicielle. Nous montrons les avantages offerts par un support de fonte rigoureux, dans le cas de trois sous-problèmes de reconnaissance : l'OCR, l'identification des fontes, et la segmentation.

[5] à une résolution donnée, p. ex. de 100 dpi.

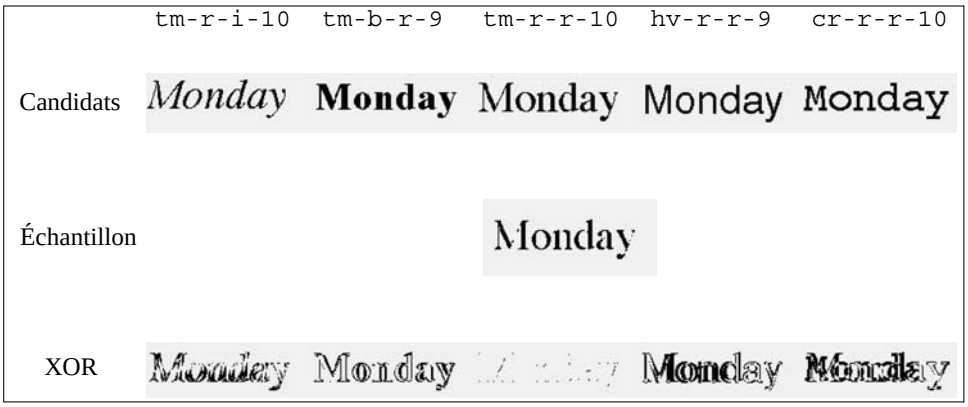


Figure 7.4 : Idée d'algorithme pour la reconnaissance de fontes a posteriori.

Contraintes typographiques

Lors du formattage d'un document, un professionnel va respecter certaines règles pour favoriser la lisibilité, la consistance, l'homogénéité ou d'autres aspects esthétiques du texte. Parmi les moyens usuels, on peut citer le placement des éléments flottants (figures, tableaux), l'ajustement des espaces inter-paragraphes, l'élimination des veuves et des orphelins (lignes isolées en fin ou début de page), et le choix systématique des fontes.

Certains travaux ont déjà mis en évidence l'utilité de ces règles utilisées en production, dans les problèmes de reconnaissance. Pour la détection des changements de fontes, Duffy et al. [63] combinent une méthode d'appariement des caractères avec des conventions sur l'usage des fontes (une fonte par mot, au plus deux fontes par suite de trois mots, au plus trois fontes par ligne...). Les défaillances locales sont surmontées par des contraintes de transitivité. Cooperman [48] décrit un sous-système de l'OCR ScanWorX [29], dédié à la reconnaissance d'attributs typographiques. Il utilise des règles empiriques. Par exemple, l'attribut italique varie souvent entre mots voisins, alors que le texte en gras est souvent localisé au début d'une phrase, et qu'on ne change habituellement pas entre chasse fixe et variable à l'intérieur d'un paragraphe.

Notons enfin que des environnements d'édition comme LaTeX [120] sont parvenus à incorporer des critères de qualité formant une véritable expertise typographique. A notre connaissance, aucune étude n'a encore établi si ces connaissances pouvaient être exploitées par un système de reconnaissance, par exemple pour valider des résultats de mise en page.

Reconnaissance de fontes

Le système *ApOFIS* développé par Zramdini [213] a démontré que la reconnaissance de fontes a priori atteint une précision remarquable. Mais l'une des difficultés de cette approche tient à la constitution de la base de connaissances. Rappelons que celle-ci se construit par apprentissage sur une portion significative d'images de texte, de l'ordre d'une page entière pour chaque fonte. Or, même dans un environnement assisté, cette tâche se révèle pénible : non seulement l'utilisateur doit isoler des échantillons de documents adéquats, mais il lui faut également nommer une nouvelle fonte par un identificateur. Compte tenu des ressemblances possibles entre plusieurs polices, rien ne garantit la consistance de cette base de données, puisque même des professionnels risquent de confondre deux polices.

Grâce à un support de fontes comme celui de X11, nous proposons d'automatiser entièrement la phase d'apprentissage, en générant des images de texte synthétiques. Il suffit de choisir les entrées de la base de connaissances parmi les fontes disponibles. Le prix à payer est une légère dégradation des performances si les échantillons à analyser sont trop dégradés [214], auquel cas on peut toujours essayer de compenser la dissymétrie par apprentissage incrémental, ou en introduisant un modèle de bruit.

Lorsque l'approche a priori s'applique mal, par exemple sur des mots isolés, le système de reconnaissance devrait pouvoir compter sur une méthode a posteriori. Ici encore, nous comptons sur un support de fontes pour simplifier énormément la mise en oeuvre d'un tel analyseur. L'algorithme de base, suggéré par la figure 7.4 et évoqué par Khoubyari et al. [107], consiste à formatter le texte d'un mot selon différentes fontes installées, générant ainsi une série d'images synthétiques. Il s'agit ensuite de calculer une mesure de dissimilarité entre ces images et l'original, par exemple par la distance de Hamming.

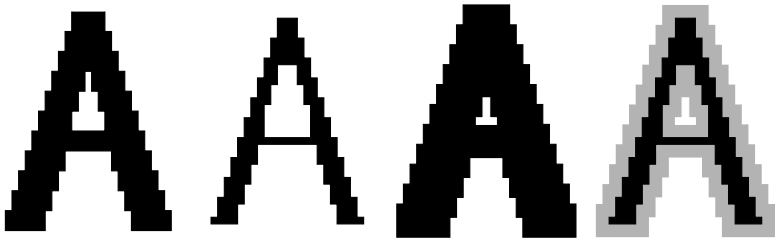


Figure 7.5 : Masques ternaires – forme idéale, squelette, enveloppe.

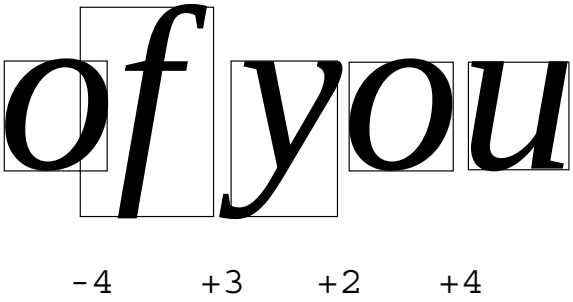


Figure 7.6 : Espacement idéal calculé avec les métriques (`tm-r-i-10`, 300 dpi).

OCR

Alors que la tendance dans le développement d'OCR s'est clairement orientée vers les technologies omnifontes, nous continuons à porter un vif intérêt à l'approche monofonte. En restreignant le problème, on s'attend en effet à améliorer les performances. Sennhauser [186] par exemple montre comment améliorer la reconnaissance de caractères grâce à des contraintes typographiques. De plus, les progrès accomplis en reconnaissance de fontes suscitent un regain d'intérêt pour l'approche monofonte.

Nous considérons que le recours à un support de fontes pour la conception d'un OCR offre deux avantages essentiels. D'une part, l'algorithme d'analyse peut tirer parti d'informations détaillées sur les formes à reconnaître et la manière de les agencer. Par exemple, nous pouvons automatiquement construire des masques ternaires [90] en appliquant des opérateurs morphologiques (érosion, dilatation) sur les formes synthétiques. La figure 7.5 illustre l'idée des masques ternaires : à partir du caractère synthétique, on génère un *squelette* et une *enveloppe* qui définissent en somme une frontière plus souple, englobant des variantes de la forme idéale. De plus, les métriques guident la reconnaissance lors de la progression dans le mot. La section 8.2 présente un algorithme complet d'OCR monofonte

D'autre part, il devient trivial de concevoir un OCR *générique*, c.-à-d. paramétré par le nom de n'importe quelle fonte installée dans le système hôte. Il est d'ailleurs surprenant qu'aucun analyseur de ce genre ne soit encore disponible dans la communauté de reconnaissance de documents, du moins à notre connaissance. Dans nos réalisations, nous avons tenté de combler cette lacune [82] (cf. section 8.2 et annexe B).

Segmentation

La segmentation en mots n'est pas un problème complètement résolu en analyse d'images de documents [176]. Pour détecter le caractère *espace*, la technique de base consiste à analyser les plages blanches au sein de la ligne de texte, en recourant à divers algorithmes : extraction des composantes connexes, RLSA, ou encore profil de projection. Le problème avec cette approche tient à l'ajustement des seuils, et la figure 7.6 illustre la principale source de difficultés : pour de nombreuses fontes, en particulier en présence d'italique, l'espacement entre caractères d'un mot peut être plus grand qu'entre deux mots, suivant les caractères impliqués.

La *connaissance préalable du texte* formant la ligne aide à compenser ces imperfections. Certains OCR commerciaux comme ScanWorX [29] exploitent par exemple des connaissances lexicales pour rejeter des frontières invalides. Toutefois, des ambiguïtés subsistent, et cette approche suppose que le texte respecte strictement le lexique.

Notre idée d'exploiter un support de fontes ouvre la voie à une autre méthode a posteriori que nous appelons *seuillage contextuel*. Le principe consiste à consulter le détail des métriques d'espacement, spécifiées par la fonte. La *table de chasse* établit la largeur des caractères. Les *approches* gauches et droites indiquent l'espacement avant et après chaque caractère. Enfin, la *table de crénage* affine le modèle pour des couples de caractères particuliers.

Supposons que les enveloppes de caractère (ou de sous-chaînes d'un mot) sont disponibles, ainsi que leur interprétation ASCII et la fonte utilisée. Dans ce cas, nous pouvons déterminer l'espacement idéal entre enveloppes consécutives, avec ou sans un séparateur espace. La décision finale découle des différences entre l'espacement observé et les valeurs idéales.

Notons encore que la fonte donne aussi une indication sur l'interligne normal, ce qui pourrait être utilisé pour guider la segmentation en lignes.

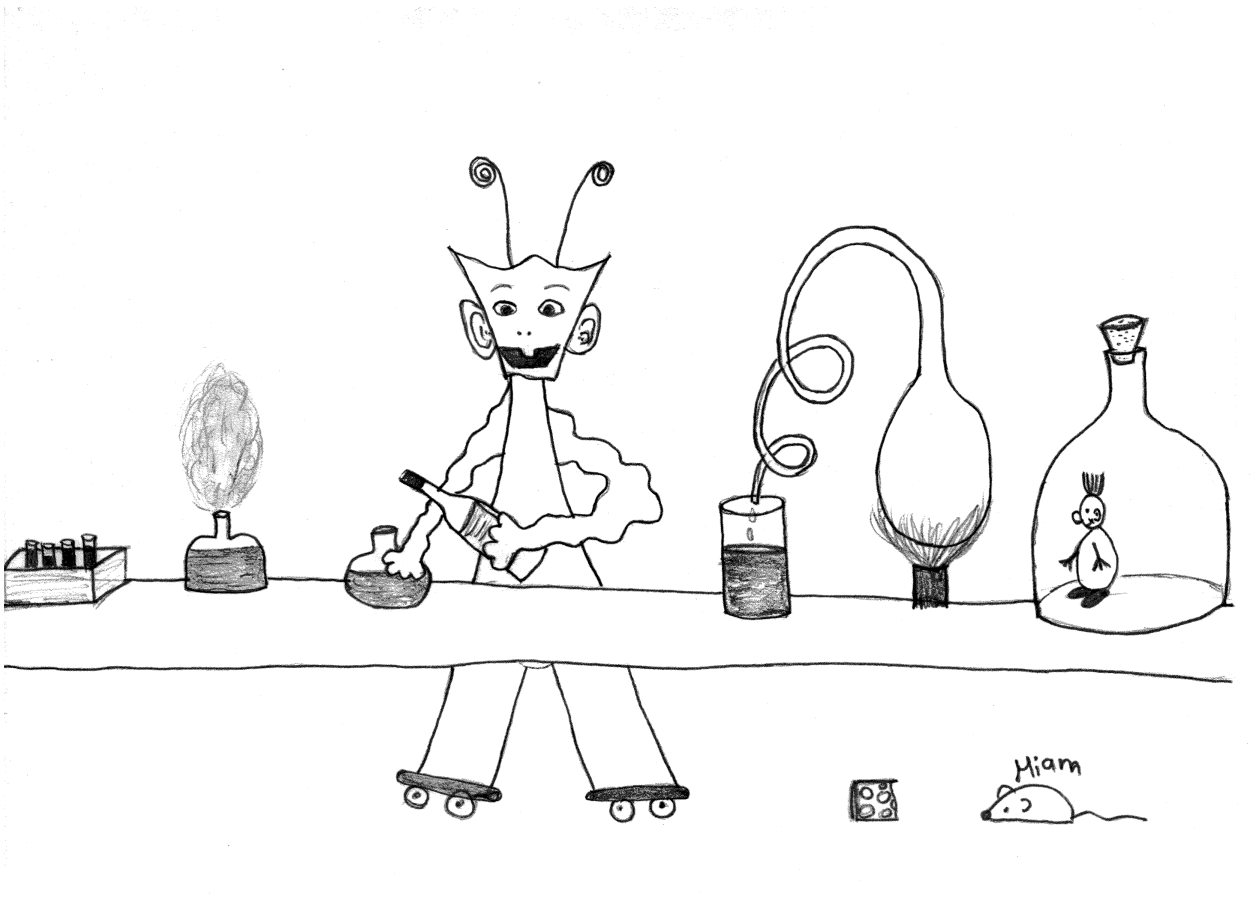
Conclusion

L'approche assistée nous conduit à réviser les principes qui gouvernent habituellement le développement d'outils d'analyse. L'étude des interdépendances entre analyseurs aide à concevoir des schémas d'analyse qui profitent au maximum des informations disponibles tout au long de la session de reconnaissance. Par ailleurs, les liens entre la reconnaissance de documents et la typographie ne sont pas encore suffisamment exploités. En particulier, l'usage d'un support logiciel standard pour gérer les fontes permettrait non seulement de développer des techniques d'analyse plus précises, mais aussi de mieux valoriser les résultats.

La prochaine section présente les analyseurs que nous avons développés, marquant ainsi la concrétisation de certaines idées évoquées jusqu'ici.

Chapitre 8

Réalisations et expériences



Mis à part son champ d'étude particulièrement large, le présent travail a également abouti à la réalisation de nouveaux outils d'analyse.

La section 8.1 explique notre manière d'aborder la partie expérimentale. Les sections 8.2 à 8.7 présentent les analyseurs que nous avons développés dans le cadre de ce travail.

8.1 Démarche expérimentale

Cette section énonce nos objectifs sur la partie expérimentale du travail, et décrit le recueil des documents utilisés, que nous avons constitué pour les tests.

8.1.1 Objectifs

La réalisation d'analyseurs n'est pas l'objectif essentiel de notre thèse. Toutefois, nous avons été amenés à réaliser quelques outils d'analyse, afin d'appuyer certaines idées que nous avons défendues jusqu'ici :

- le recours à un support logiciel standard pour gérer les fontes (cf. sections 8.2, 8.3 et 8.4);

- la possibilité de répercuter automatiquement la correction d'un résultat d'OCR (cf. section 8.6);
- la conduite non dirigiste du dialogue, en particulier dans les fonctionnalités d'apprentissage incrémental (cf. section 8.5);
- la reconnaissance de documents électroniques, en tant qu'application particulière de l'analyse d'images de documents (cf. section 8.7).

Les techniques d'analyse que nous avons utilisées ne sont pas révolutionnaires. Les algorithmes sont même assez simples, suivant en cela l'argumentation présentée dans la section 7.1.1. Notre principal souci n'a pas été d'optimiser la vitesse ni la précision, mais de prévoir l'intégration dans le reste de l'architecture logicielle. Sur le plan de l'implémentation, le format DAFS [174] a été utilisé aussi souvent que possible.

Les expériences que nous avons menées donnent des indications intéressantes sur l'utilité potentielle de nos modules. En revanche, nous n'avons pas encore entrepris une évaluation à large échelle, qui permettrait de comparer les résultats avec les techniques rapportées dans la littérature.

8.1.2 Base de données pour les tests

La pertinence de l'évaluation d'un analyseur tient notamment au choix des données utilisées dans les expériences. Dans ce contexte, des standards se mettent peu à peu en place. Nous avons ainsi installé la base de données d'images de documents UW, produite par l'Université de Washington [127]. Certains travaux [61, 13] ont étudié la possibilité de produire des grands volumes de données adaptés à la reconnaissance de caractères.

Toutefois, nos analyseurs reposent sur des attributs précis, comme la description rigoureuse de la fonte, ou l'enveloppe des caractères. Pour estimer la précision des résultats, nous avons besoin de documents correctement reconnus jusque dans ces détails. Or, la base de données UW [168] présente pour le moment trois défauts : (i) elle n'est pas encore indexée avec des attributs typographiques suffisamment précis, (ii) la segmentation s'arrête au niveau du mot, et (iii) le texte correct est associé seulement au niveau des paragraphes.

C'est pourquoi nous avons conçu notre propre jeu de données, destiné à refléter la variété des entités textuelles. Chaque document représente l'image d'une page A4 à une résolution de 300 dpi, contenant un unique texte en anglais[1]. Les divers documents de la base de données varient en fonction de plusieurs propriétés :

- la fonte utilisée pour formatter le texte : la base contient 390 fontes, qui représentent les 19 polices installées dans notre architecture, six tailles (8, 9, 10, 11, 12, 14pt), ainsi que, si la fonte le permet, quatre styles (romain/italique, régulier/gras);
- le mode de génération des pages : la base contient des pages générées soit directement avec X11, soit avec LaTeX (mais seulement pour 174 fontes);
- le mode d'acquisition des images : la base contient des documents imprimés puis scannés, ainsi que des images synthétiques;
- le niveau de dégradation : pour simuler quatre niveaux de dégradation, nous avons utilisé l'utilitaire DDM [102] (Document Degradation Model), distribué avec la base de données UW.

Pour chaque document, nous avons construit une solution idéale, dans un processus quasi automatique. La figure 8.1 présente la moitié supérieure d'une page typique de notre base de données. La figure 8.2 illustre la qualité originale et trois niveaux de dégradation, sur un document LaTeX scanné (haut), et sur un document X11 synthétique (milieu) ou scanné (bas). Suivant la taille de la fonte, chaque document contient environ entre 500 et 1000 mots, soit entre 3000 et 6000 caractères.

Cet ensemble de données nous semble raisonnable. En particulier, il était important d'incorporer des documents LaTeX, qui sont formattés avec un autre support de fontes que celui qu'utilise notre plateforme de reconnaissance. Nous avons certes trouvé une correspondance avec les fontes installées sur notre système X11, mais ce n'est pas une exacte copie des polices installées sous LaTeX. D'un point de vue *méthodologique*, les analyseurs qui sont paramétrés par une fonte doivent être testés sur du texte produit avec exactement la même fonte. D'un point de vue *pragmatique* en revanche, il est surtout utile de montrer le comportement de l'outil dans une situation plus réaliste, à savoir celle où on doit se contenter de choisir la fonte la plus proche.

[1] pour l'évaluation de certains analyseurs, il pourrait être pertinent de choisir un texte contenant toutes les combinaisons de couples (ou de n-uplets) de caractères.

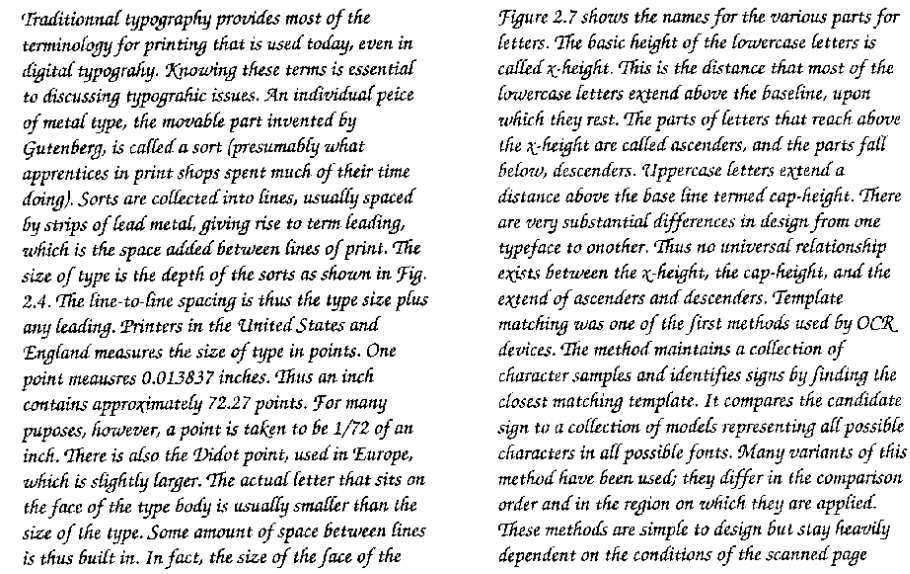


Figure 8.1 : Image de notre base de données typographique (`zc-r-r-12`).

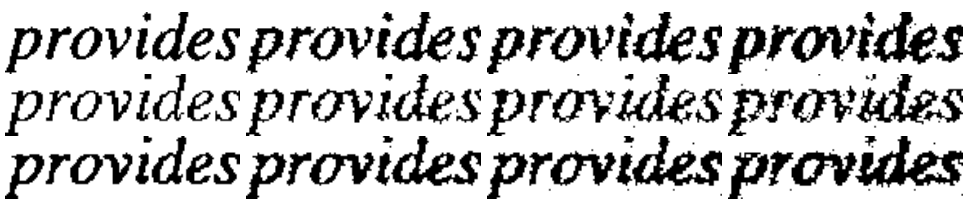


Figure 8.2 : Dégradation artificielle (documents LaTeX, X11 synthétiques, et X11 scannés).

8.2 OCR monofonte générique

Cette section présente les caractéristiques principales de notre OCR monofonte générique, que nous avons développé au cours du projet *CIDRE*. Une partie importante du cahier des charges a été confiée à Oliver Hitz, dans le cadre de son travail de diplôme [82].

8.2.1 Objectifs

Malgré des décennies d'études intensives, la reconnaissance de texte imprimé n'est pas encore complétement résolue [146]. Nous voyons surtout cinq sources de difficultés :

- Il s'est avéré impossible de concevoir un analyseur qui reconnaisse *n'importe quelle fonte*. En ce sens, l'approche omni-fonte aboutit à une impasse, et se limite de fait à un sous-ensemble de fontes «classiques». Les contraintes typographiques sont donc importantes [15, 186].
- La *segmentation en caractères* est loin d'être une étape fiable, notamment à cause de caractères séparés ou fusionnés.
- Le *crénage* pose non seulement des problèmes de segmentation, mais aussi de reconnaissance, car il entrave la détection des pixels significatifs autour des caractères présents dans l'image.
- Les images contiennent du *bruit*. Parmi les différentes formes de bruits, il y a l'effet de discrétisation [182, 132] dû à la saisie au scanner.
- La précision des résultats se dégrade en l'*absence de contraintes lexicales*.

Par ailleurs, il nous semble qu'aucun paquetage ne peut actuellement être considéré comme une base solide pour les projets de reconnaissance de documents. Les rares outils du domaine public sont de piètre qualité[2], et les produits commerciaux, sur-optimisés pour les usages les plus courants, sont difficiles à ajuster en vue d'une utilisation non standard.

[2] on attendra quand même la prochaine version de SOCR [89].

Figure 8.3 : Exemple de masque produit par notre méthode.

Face à cette situation, nous avons voulu développer notre propre outil de reconnaissance de caractères imprimés, en respectant les principes suivants :

- adopter une approche monofonte;
- concevoir un analyseur générique, c.-à-d. que l'on peut facilement reconfigurer pour n'importe quelle fonte;
- exploiter le support de fontes offert par l'environnement informatique, en l'occurrence les fonctionnalités de X11 (cf. section 7.3);
- utiliser une technique de comparaison directe d'images binaires, et renoncer à l'extraction de caractéristiques complexes, ou à l'usage de connaissances lexicales;
- favoriser au maximum la simplicité de l'algorithme, afin de pouvoir facilement étendre l'analyseur.

En ce qui concerne la segmentation en caractères, quatre approches se sont imposées. La première cherche à segmenter en caractères, en utilisant des techniques d'analyse avancées [156, 129]. La deuxième vise à reconnaître en bloc les caractères consécutifs quand les frontières sont douteuses, en ajoutant à l'alphabet de base certaines combinaisons de caractères (comme 'fi'). La troisième consiste à identifier le mot entier [163], et suppose que le lexique est connu. La quatrième consiste à reconnaître les caractères dans le contexte, en optimisant la mise en correspondance par une analyse de tout le mot. C'est cette dernière approche que nous avons adoptée.

8.2.2 Génération de masques

Notre approche consiste à construire un masque pour chaque caractère, dans le but de procéder à une reconnaissance dans le contexte. Le masque doit tenir compte de la forme idéale du caractère, mais aussi des informations sur le contexte possible, c.-à-d. des règles d'assemblage du caractère avec tout l'alphabet. Nous utilisons le concept de *masque ternaire* [90], où la surface d'un caractère est définie par deux ensembles de pixels : le *squelette* représente l'ensemble des pixels qui doivent être noirs, et l'*enveloppe* ceux qui doivent être blancs.

Nous utilisons la description des fontes offerte par X11 pour construire les masques ternaires, par dérivation de la forme idéale de caractères. De plus, nous calculons la superposition de tous les caractères lorsqu'ils (i) précèdent ou (ii) suivent un point d'ancrage. Ces informations sont rajoutées à l'enveloppe de chaque caractère. De cette manière, les masques anticipent l'effet contextuel, p. ex. dû au crénage. Les masques idéaux seraient trop sensibles au bruit. Afin d'anticiper l'effet de discrétisation, nous appliquons deux opérations morphologiques sur les masques, soit une érosion sur le squelette et une dilatation sur l'enveloppe.

La figure 8.3 présente un masque généré par notre méthode. Le squelette (au centre) correspond à une version érodée de la forme idéale (à gauche). Dans l'enveloppe (à droite), on remarque les bords noirs non significatifs, qui correspondent aux pixels pouvant appartenir à un caractère voisin.

8.2.3 Algorithme de reconnaissance

Nous supposons connus le rectangle délimitant le mot à reconnaître, la fonte et la hauteur de la ligne de base. L'algorithme consiste à progresser itérativement dans le mot, en avançant le point d'ancrage après chaque

caractère reconnnu. A chaque position du point d'ancrage, nous testons tous les caractères de l'alphabet, puis nous les trions selon leur valeur de dissemblance.

L'algorithme tolère quelques pixels de décalage horizontal ($\pm d_x$) et vertical ($\pm d_y$) par rapport au point d'ancrage, et ne conserve pour chaque candidat que le meilleur score. L'avance du point d'ancrage est déterminée par les métriques de la fonte.

Un aspect important de notre algorithme réside dans le procédé de décision. Lorsqu'il y a peu d'écart entre les meilleurs candidats à une position donnée, nous retenons *plusieurs hypothèses* pour explorer la suite du mot. De cette manière, notre algorithme évalue un ensemble de chemins alternatifs, et choisit le meilleur résultat seulement à la fin du mot. Cette méthode tire profit du phénomène suivant, valable pour les fontes à chasse variable. Lorsqu'on choisit le mauvais candidat à une étape donnée (p. ex. un 'I' au lieu d'un 'K'), il arrive souvent que la mise en correspondance soit nettement moins bonne sur la suite du mot, parce que le point d'ancrage n'est plus synchronisé avec le début des caractères. C'est aussi une réponse au problème de la différence d'échelle entre les petits caractères ('.,; ') et les grands (Wm&).

Pour comparer les masques avec l'image du mot, nous avons choisi la *fonction de dissemblance* suivante[3]. Soient S et E le nombre de pixels appartenant au squelette, respectivement à l'enveloppe. Soient A et B le nombre de pixels dans l'image qui sont en désaccord avec le squelettte, respectivement l'enveloppe. Nous calculons une somme pondérée D des pixels dissemblables au moyen de la formule :

$$D = \frac{S+E}{2}\left(\frac{A}{S} + \frac{B}{E}\right) \tag{8.1}$$

Cette mesure offre les avantages suivants :

- elle est peu biaisée par la qualité des masques : la mesure rétablit l'équilibre entre l'apport du squelette et celui de l'enveloppe, même lorsqu'il y a bien davantage de pixels significatifs dans l'un que dans l'autre;
- elle est comprise entre 0 et $S + D$, et peut donc être normalisée par une division;
- elle permet d'évaluer aussi bien un seul caractère que tout un chemin dans le mot.

8.2.4 Optimisations

Telle que nous l'avons présentée, notre méthode bute sur un phénomène d'*explosion combinatoire*. Si on admet un alphabet de 100 caractères, et un décalage possible de ± 2 pixels horizontalement et verticalement, la reconnaissance d'un mot de huit lettres implique au moins 20'000 comparaisons de masques. Pour une fonte de 8pt à 300 dpi, où la surface moyenne d'un caractère serait de 100 pixels, cela représenterait environ 2 millions d'opérations, ce qui est déjà très gourmand. De plus, la conservation de chemins alternatifs dans l'analyse du mot introduit un risque de faire exploser la complexité.

Nous avons mis en oeuvre trois mécanismes d'optimisation, avec l'objectif prioritaire de ne pas entraver la simplicité de l'algorithme.

- *Présélection :* avant de balayer l'image pour comparer toute la surface d'un masque, nous effectuons une présélection en ne comparant que la ligne et la colonne centrales du masque, comme illustré sur la figure 8.4. Nous abandonnons les caractères où la dissemblance normalisée dépasse un certain seuil α. Dans nos tests, environ 95% des comparaisons ont pu être évitées par ce filtre.
- *Mort subite :* après le balayage de chaque colonne, nous consultons le nombre cumulé de pixels dissemblables. Si cette valeur dépasse une proportion β de tous les pixels significatifs du masque, le candidat est abandonné. A nouveau, nos tests ont montré qu'environ 95% des comparaisons étaient écourtées par ce procédé.
- *Réduction des chemins :* la création d'un chemin alternatif est limité par le respect de deux règles : (i) l'écart entre le score du caractère testé et celui du meilleur candidat doit être inférieur à un certain pourcentage γ; (ii) le nombre maximal d'alternatives locales ne doit pas dépasser une constante δ.

Les performances sont influencées par les *facteurs de complexité* suivants :

- la taille de l'alphabet utilisé;
- la longueur du mot à reconnaître;

[3] voir aussi [61] pour des réflexions intéressantes sur les mesures de dissemblance entre images de caractères.

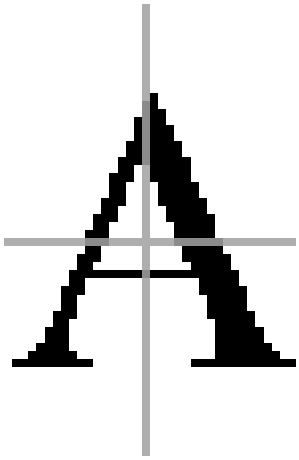


Figure 8.4 : Présélection par un masque en croix.

- la résolution des images;
- la taille moyenne des caractères de la fonte, dépendant notamment de la taille nominale de la fonte (p. ex. 10pt);
- la séparabilité intrinsèque des classes dans l'alphabet; les calculs augmentent si certains caractères se ressemblent beaucoup[4], ou si certains caractères sont le préfixe d'un autre (comme l'apostrophe et le guillemet).

En outre, le choix des seuils $(\alpha, \beta, \gamma, \delta)$ est déterminant pour accélérer notre algorithme.

8.2.5 Expériences

L'évaluation des résultats d'OCR [147] est facilitée par l'existence d'outils standards pour calculer des distances d'édition de chaînes, entre une interprétation proposée et le texte correct. Nous avons ainsi utilisé l'utilitaire OPE (OCR Performance Evaluation) [43] distribué avec la base de données UW.

Nous présentons ici quelques résultats obtenus sur notre base de données typographique. Dans ces tests, l'alphabet comprenait tous les caractères imprimables de la fonte, soit environ 180. L'érosion et la dilatation ont été effectuées par un masque en croix de taille 3x3, sauf pour les fontes jusqu'à 10pt, où nous n'avons pas appliqué d'érosion. Les seuils ont été choisis comme compromis entre une analyse superficielle et une analyse exhaustive. Les tests ont été menés avec rigueur. Nous avons notamment reconcé à :

- réduire l'alphabet aux seuls caractères présents dans le document;
- réordonner les caractères dans l'alphabet;
- choisir pour chaque fonte la meilleure combinaison d'érosion/dilatation;
- relancer une analyse exhaustive sur les mots rejetés.

Le tableau 8.1 présente les taux de précision obtenus sur les documents X11 synthétiques, de taille 12pt et 14pt, avec ou sans le premier degré de dégradation.

Les résultats semblent excellents. Seules les fontes sans sérifs sont sensiblement moins bien reconnues. A y regarder de plus près, la plupart de ces erreurs sont issues d'une confusion entre '`l`' minuscule et '`I`' majuscule. D'après la fonte, ces deux formes sont strictement identiques. Dans le texte analysé, il y a beaucoup plus de '`l`'. Or, le '`I`' apparaît en premier dans notre alphabet, et est donc privilégié par notre algorithme. D'après notre hypothèse de ne recourir qu'à l'image, on pourrait estimer que ce ne sont pas vraiment des erreurs.

Il est cependant assez risqué de se baser uniquement sur des documents synthétiques dégradés artificiellement pour prédire le comportement d'un analyseur dans des situations réelles. C'est pourquoi l'annexe B rapporte d'autres expériences avec notre OCR, portant sur des pages scannées.

[4] si un seul chemin est poursuivi, nous avons constaté qu'il était plus précis de parcourir le mot de droite à gauche, probablement parce que dans le dessin courant de l'alphabet latin, il y a plus de préfixes communs que de suffixes communs.

	Sans dégradation		Avec dégradation			Sans dégradation		Avec dégradation	
Fonte	12pt	14pt	12pt	14pt	Fonte	12pt	14pt	12pt	14pt
ag-r-r	0.971	0.971	0.994	0.993	hvn-r-r	0.991	0.995	1.000	0.995
ag-r-i	0.971	0.971	0.994	0.996	hvn-r-i	0.993	0.983	0.996	0.995
ag-b-r	0.994	0.989	0.995	1.000	hvn-b-r	0.968	0.966	0.983	1.000
ag-b-i	0.989	0.993	0.998	1.000	hvn-b-i	0.968	0.966	0.994	0.996
bem-r-r	1.000	1.000	0.998	0.998	lb-r-r	1.000	1.000	1.000	1.000
bem-r-i	1.000	1.000	0.994	0.999	lb-r-i	1.000	1.000	0.998	1.000
bem-b-r	1.000	1.000	1.000	1.000	lb-b-*	1.000	1.000	1.000	1.000
bem-b-i	1.000	1.000	0.999	1.000	ls-r-r	1.000	1.000	0.999	1.000
cm-r-r	0.989	0.987	0.986	0.990	ls-r-i	1.000	1.000	1.000	1.000
cm-r-i	1.000	1.000	0.965	0.992	ls-b-*	1.000	1.000	1.000	1.000
cm-b-r	0.989	1.000	0.991	1.000	lst-r-r	1.000	1.000	1.000	1.000
cm-b-i	1.000	1.000	1.000	1.000	lst-b-r	1.000	1.000	1.000	1.000
cr-r-r	0.998	0.999	0.991	0.999	nc-*-*	1.000	1.000	1.000	1.000
cr-r-i	0.998	0.999	0.979	0.999	pl-*-*	1.000	1.000	1.000	1.000
cr-b-r	0.998	0.999	0.998	0.999	rkw-*-*	1.000	1.000	1.000	1.000
cr-b-i	0.998	0.999	1.000	0.999	sym-r-r	0.995	0.994	0.994	0.994
gis-r-r	0.972	0.972	0.995	1.000	tm-r-r	1.000	1.000	0.999	1.000
gis-r-i	1.000	1.000	0.999	1.000	tm-r-i	1.000	1.000	0.999	1.000
gis-b-r	1.000	0.993	1.000	1.000	tm-b-*	1.000	1.000	1.000	1.000
gis-b-i	0.995	0.994	1.000	1.000	uto-r-r	1.000	1.000	1.000	1.000
hv-r-r	0.995	0.994	1.000	0.997	uto-r-i	0.999	1.000	0.999	1.000
hv-r-i	0.994	0.994	0.999	0.998	uto-b-*	1.000	1.000	1.000	1.000
hv-b-r	0.965	0.965	0.965	0.965	zc-r-r	0.999	0.998	0.962	0.957
hv-b-i	0.965	0.965	0.965	0.965	zd-r-r	1.000	1.000	1.000	1.000

Tableau 8.1 : Précision de l'OCR monofonte.

8.2.6 Perspectives

Nous sommes pleinement satisfaits de notre paquetage d'OCR monofonte. Nous avons la conviction du bien-fondé de notre approche monofonte, basée sur un support de fontes et sur une reconnaissance dans le contexte. L'utilité potentielle de notre librairie ne fait aucun doute. Les résultats sont de bonne qualité, et le débit moyen (quelques dizaines de caractères par seconde), quoiqu'un peu faible, reste raisonnable. Surtout, la simplicité de la version actuelle permet d'envisager avec confiance la mise en oeuvre de certaines extensions.

Il devrait être possible d'incorporer des *connaissances lexicales* ou linguistiques dans l'algorithme de base. Par exemple, on pourrait associer une probabilité a priori sur les caractères, ou mieux, sur un ensemble de n-grams. Il suffirait ensuite de modifier la mesure de dissemblance pour tenir compte de ces connaissances. Compte tenu des performances actuelles, nous postulons que cette extension amènerait notre analyseur à rivaliser avec les meilleurs OCR actuels pour le traitement de textes écrits dans une langue et une fonte connues.

On peut imaginer plusieurs formes d'*adaptabilité*. En effet, notre algorithme utilise une description idéale des fontes, qui ne correspond pas forcément aux caractéristiques du document traité. Suivant la fonderie, le dessin des caractères varie entre fontes du même nom. Les métriques ont pu être court-circuitées par le formatteur de texte, qui peut utiliser des règles typographiques plus évoluées, comme le fait LaTeX. Enfin, le type de dégradation subie par les images de documents (p. ex. photocopie) peut entraîner une déformation homogène du dessin des caractères dans tout le document.

On pourrait donc affiner les connaissances sur la fonte pour mieux s'adapter à un type de document. En consultant les décalages horizontaux choisis par rapport au points d'ancrage idéal, on peut reconstituer une table de crénage. Kopec [111] montre comment estimer les métriques d'une fontes. Quant aux masques des caractères, il est également possible de les ajuster par apprentissage incrémental. Par exemple, connaissant le texte d'un mot, on établit la mise en correspondance d'après les masques idéaux, puis on modifie les squelettes et les enveloppes par des opérations binaires ET/OU avec l'image analysée. Notons aussi qu'il existe des techniques pour extraire automatiquement des prototypes de caractères [154, 155]. Hobby et al. [85] montrent comment régénérer un prototype de caractère à haute résolution à partir d'échantillons à basse résolution.

Globalement, nous avons l'impression d'avoir réalisé un outil très utile pour la reconnaissance du texte imprimé.

Fonte		Fonte		Fonte		Fonte	
ag-r-r-10	0.97	nc-r-r-10	0.95	lb-r-r-10	0.50	tm-r-r-10	0.60
ag-r-i-10	0.98	nc-r-i-10	0.98	lb-r-i-10	0.86	tm-r-i-10	0.60
ag-b-r-10	0.99	nc-b-r-10	0.99	lb-b-r-10	0.90	tm-b-r-10	0.62
ag-b-i-10	0.99	nc-b-i-10	0.99	lb-b-i-10	0.96	tm-b-i-10	0.60
cr-r-r-10	0.95	pl-r-r-10	0.72	hv-r-r-10	0.48	zc-r-r-10	0.83
cr-r-i-10	0.98	pl-r-i-10	0.90	hv-r-i-10	0.97		
cr-b-r-10	0.98	pl-b-r-10	0.99	hv-b-r-10	0.78		
cr-b-i-10	0.99	pl-b-i-10	0.99	hv-b-i-10	0.97		

Tableau 8.2 : Précision de la reconnaissance de fontes a posteriori (doc. LaTeX).

8.3 Reconnaissance de fontes

8.3.1 Méthodologie

L'idée de notre reconnaissance de fontes a posteriori consiste à faire correspondre l'image à identifier avec des versions synthétiques du même mot dans différentes fontes (cf. section 7.3).

En pratique, notre implémentation repose sur notre OCR monofonte (cf. annexe B). Nous avons défini une routine qui optimise la mise en correspondance de chaque caractère du mot avec l'image analysée. Après avoir positionné finement chaque caractère, nous progressons dans l'image du mot, en tenant compte des métriques du caractère. En ce sens, l'algorithme est équivalent à celui de la reconnaissance de caractères, à la différence qu'à chaque étape de la progression dans le mot, seul le bon caractère est testé. Les temps de calcul sont ainsi considérablement réduits.

Sur chaque mot, cette routine est appelée avec chacune des fontes candidates. La méthode d'OCR retourne une mesure des pixels qui sont en désaccord avec les masques des caractères reconnus. Ces scores permettent de classer ensuite les fontes candidates, afin de déterminer la fonte la plus proche.

8.3.2 Expériences

Notre analyseur a été testé sur les 174 documents LaTeX de notre base de données (scannés, sans dégradation), ainsi que sur les documents X11 correspondants (synthétiques, avec le premier niveau de dégradation) Pour chaque échantillon analysé, l'analyseur devait déterminer la fonte de chaque mot individuellement, parmi les 174 candidats.

En fait, il s'agit d'un scénario très défavorable à plus d'un titre. D'une part, l'ensemble des candidats est très grand, alors que pour la plupart des applications, on peut limiter sensiblement le nombre de fontes attendues dans un type de documents. D'autre part, des contraintes d'homogénéité apporteraient une amélioration notable dans le traitement de mots consécutifs. Si on se trompe sur un mot avec une probabilité p (p petit), alors le taux d'erreurs sur n mots tend rapidement vers 0 lorsque n croît, pour autant que les erreurs soient indépendantes.

Le tableau 8.2 présente les taux de reconnaissance pour la taille 10pt, sur les documents LaTeX scannés. Les résultats montrent que la méthode est plutôt précise, compte tenu des conditions extrêmement défavorables du test : 174 fontes candidates, plusieurs tailles très proches, analyse mot par mot sans contraintes d'homogénéité, fontes X11 au lieu des fontes originales LaTeX. Pour plus de la moitié des fontes, la précision dépasse pourtant 95%.

Comme prévu, les confusions concernent des fontes très similaires. Un examen attentif des pires cas a révélé certaines différences entre les polices de production et de reconnaissance. Pour les fontes `hv-r-r-10` et `tm-b-i-10` par exemple, notre analyseur a reconnu correctement la famille et le style, mais a confondu avec la taille 11pt. En outre, cette expérience a permis de vérifier que la fonte la plus proche de chaque police LaTeX était bien son équivalent X11, ce qui justifie pleinement notre approximation.

Le tableau 8.3 rapporte la précision des résultats sur les documents X11 scannés, sans dégradation artificielle. Les résultats sont excellents, à l'exception d'une fonte Palatino, qui a été souvent confondue avec sa version Bold.

Il est tentant de comparer notre analyseur avec l'outil *ApOFIS* [213], même si les approches sont diamétralement opposées. Rappelons qu'*ApOFIS* adopte une approche a priori (sans connaissance du texte). Zramdini [213] rapporte que sous des conditions d'apprentissage adéquates, *ApOFIS* a reconnu la fonte

Fonte		Fonte		Fonte		Fonte	
ag-r-r-10	0.99	nc-r-r-10	0.93	lb-r-r-10	0.88	tm-r-r-10	0.96
ag-r-i-10	1.00	nc-r-i-10	0.99	lb-r-i-10	0.95	tm-r-i-10	0.98
ag-b-r-10	1.00	nc-b-r-10	0.99	lb-b-r-10	0.92	tm-b-r-10	0.96
ag-b-i-10	1.00	nc-b-i-10	0.98	lb-b-i-10	0.98	tm-b-i-10	1.00
cr-r-r-10	0.90	pl-r-r-10	0.81	hv-r-r-10	0.98	zc-r-r-10	0.99
cr-r-i-10	0.99	pl-r-i-10	0.98	hv-r-i-10	0.98		
cr-b-r-10	0.99	pl-b-r-10	0.99	hv-b-r-10	0.99		
cr-b-i-10	0.99	pl-b-i-10	1.00	hv-b-i-10	1.00		

Tableau 8.3 : Précision de la reconnaissance de fontes a posteriori (doc. X11).

Fonte		Fonte		Fonte		Fonte	
ag-r-r-10	0.65	nc-r-r-10	0.78	lb-r-r-10	0.51	tm-r-r-10	0.89
ag-r-i-10	0.85	nc-r-i-10	0.87	lb-r-i-10	0.62	tm-r-i-10	0.48
ag-b-r-10	0.86	nc-b-r-10	0.86	lb-b-r-10	0.87	tm-b-r-10	0.61
ag-b-i-10	0.91	nc-b-i-10	0.92	lb-b-i-10	0.89	tm-b-i-10	0.77
cr-r-r-10	0.93	pl-r-r-10	0.80	hv-r-r-10	0.78	zc-r-r-10	0.41
cr-r-i-10	0.91	pl-r-i-10	0.87	hv-r-i-10	0.89		
cr-b-r-10	0.90	pl-b-r-10	0.78	hv-b-r-10	0.90		
cr-b-i-10	0.89	pl-b-i-10	0.83	hv-b-i-10	0.82		

Tableau 8.4 : Précision de la reconnaissance de fontes a priori (doc. X11).

d'une ligne avec une précision de l'ordre de 95.8%, dans des expériences similaires aux nôtres sur le plan des images analysées et de l'ensemble de fontes candidates. Pour compléter cette mesure de précision, nous avons cherché à appliqué *ApOFIS* au niveau du mot. Dans cette expérience, nous avons réduit la difficulté de trois manières : (i) on se limite aux images synthétiques non bruitées; (ii) les mêmes échantillons de mots sont utilisés pour l'apprentissage et la reconnaissance; (iii) seules les tailles 8, 10, 12, et 14pt sont incluses dans les fontes candidates. Le tableau 8.4 indique la précision observée pour la taille 10pt. Même avec les simplifications consenties, la précision reste sensiblement inférieure à notre algorithme. Toutefois, il convient de garder à l'esprit que notre approche suppose connu le texte de chaque mot, ce qui est une forte hypothèse. Par ailleurs, il serait intéressant de comparer le comportement des deux analyseurs en fonction du niveau de dégradation des images.

8.4 Segmentation en mots

8.4.1 Méthodologie

Afin de réduire les erreurs de segmentation en mots commises par les techniques usuelles (cf. section 7.3.3), nous avons proposé une méthode de *seuillage contextuel*. L'idée consiste à consulter les métriques de la fonte. Il s'agit d'une approche a posteriori, dans la mesure où elle exploite la connaissance préalable de la fonte ainsi que des caractères présents dans une ligne. Précisons encore que c'est la détection des séparateurs espace qui nous intéresse, et pas la présence de mots au sens linguistique.

La figure 8.5 décrit l'algorithme sous forme de pseudo-code. Les tableaux `b` et `s` caractérisent le texte reconnu (enveloppes rectangulaires et interprétation ASCII), décomposé en `n` groupes de un ou plusieurs caractères (p. ex. collés). Ce format est tout à fait compatible avec les résultats d'un OCR comme ScanWorX [29]. L'algorithme retourne les séparateurs espace dans le tableau `<isSpaceBefore>`.

8.4.2 Expériences

Mesurer la qualité de la segmentation en mots n'est pas trivial. Il s'agit de comparer les résultats avec une version correctement segmentée, en mettant en correspondance les séparateurs de mots, en fonction de leur position dans l'image. Nous distinguons deux types d'erreurs, suivant qu'un espace manque ou qu'il a été rajouté. Les pourcentages indiqués par la suite se rapportent à la proportion de séparateurs correctement détectés.

Nous avons testé notre analyseur sur les documents LaTeX scannés (174 fontes, images sans dégradation),

```
PROC DetectSpacesInLine(b: ARRAY OF Box;
                        s: ARRAY OF String;
                        n: Integer;
                        f: Font;
                    VAR isSpaceBefore : ARRAY OF BOOLEAN);
VAR crtChar, nxtChar:  Char;
    i, crtX, observedOffset, idealOffset: Integer;
BEGIN
  isSpaceBefore[0]:= FALSE;
  crtX:= b[0].x + b[0].w;
  crtChar:= LastChar(s[0]);
  FOR i:=1 TO n-1 DO
    observedOffset:= b[i].x - crtX;
    nxtChar:= FirstChar(s[i]);
    idealOffset:= (CharWidth(f, crtChar) - RightBearing(f, crtChar))
               + LeftBearing(f, nxtChar);
    isSpaceBefore[i]:= (observedOffset-idealOffset > CharWidth(f,' ')/2);
    crtX:= b[i].x + b[i].w;
    crtChar:= LastChar(s[i]);
  END;
END;
```

Figure 8.5 : Pseudo-code pour le seuillage contextuel.

et correctement reconnus au niveau caractère. Puis nous avons cherché à comparer notre analyseur avec les logiciels d'OCR actuels. Le tableau 8.5 rapporte la précision de ces deux outils dans nos expériences. On constate que ScanWorX commet parfois un nombre d'erreurs considérable sur la segmentation en mots [176]. Pourtant, il faut signaler que ce logiciel surmonte la plupart des difficultés grâce à ses connaissances lexicales. La précision se dégraderait encore drastiquement sur du texte aléatoire.

Globalement et à trois petites exceptions près, la précision est supérieure pour notre méthode, qui ne consulte pas de lexique mais utilise les métriques de la fonte. Un examen attentif de ces erreurs a révélé que LaTeX effectue souvent un crénage plus marqué que X11, surtout autour du *'f'* italique.

8.5 Module d'apprentissage pour ScanWorX

8.5.1 Méthodologie

Le logiciel d'OCR ScanWorX [29] est un produit commercial développé par Xerox, dont les performances passent pour être plutôt bonnes [176]. En plus d'un lexique propre à l'utilisateur, cet analyseur supporte une véritable forme d'apprentissage incrémental. La figure 8.6 illustre le recours à cette fonctionnalité. L'utilisateur entre dans un mode spécial, et appelle l'analyseur pour reconnaître une portion de texte. Ensuite, à chaque fois que le logiciel a un doute sur l'interprétation d'un glyphe, il se bloque et sollicite l'expertise humaine. L'opérateur dispose alors d'un ensemble de commandes pour valider, ignorer, fusionner des composantes connexes, ou modifier le texte proposé. Ces connaissances sont accumulées dans un fichier d'apprentissage, qui sera réutilisé pour les futures phases de reconnaissance automatique.

Il s'agit donc d'un mode d'interaction dirigiste, particulièrement pénible à subir. En effet, les situations douteuses se révèlent très nombreuses, et l'opérateur perd son temps à valider des résultats corrects. Ainsi, bien que l'outil soit à la fois adaptatif et interactif, nous estimons qu'il manque de souplesse en vue d'un environnement convivial. A notre avis, la visualisation et les corrections doivent être conduites librement par l'opérateur. Une fois qu'une portion de texte a été corrigée, l'apprentissage devient une opération automatique, qui n'a plus besoin d'interagir avec l'utilisateur, puisque l'expertise de ce dernier est accessible par l'intermédiaire du texte validé.

Nous avons donc développé un module qui prend en entrée une portion d'image et l'interprétation textuelle exacte, qui lance une session d'apprentissage, et interagit automatiquement en respectant le protocole de ScanWorX. Le principe d'exécution de ce module est schématisé sur la figure 8.7. L'algorithme suppose que l'on dispose d'une description détaillée de la solution correcte, jusqu'au positionnement exact de chacun des caractères. En pratique, si on a à disposition l'enveloppe du mot dans l'image, la chaîne de caractères, la

Fonte	(a)	(b)	Fonte	(a)	(b)	Fonte	(a)	(b)
ag-b-i-8	99.9	100	ag-r-i-10	99.8	100	ag-r-i-14	99.6	100
ag-r-i-8	99.7	100	hv-b-i-14	99.6	100	hv-r-i-14	99.8	100
hv-r-i-8	99.9	100	lb-b-i-8	99.8	100	lb-r-i-10	99.8	100
lb-r-i-11	99.6	100	lb-r-i-12	99.7	100	lb-r-i-14	99.6	100
lb-r-i-8	99.4	100	lb-r-i-9	99.7	100	nc-b-i-10	99.3	100
nc-b-i-11	98.7	100	nc-b-i-12	99.7	100	nc-b-i-14	99.6	100
nc-b-i-8	98.6	100	nc-b-i-9	99.6	100	nc-r-i-11	99.5	100
nc-r-i-8	99.7	100	nc-r-i-9	99.8	100	pl-b-i-10	99.3	100
pl-b-i-11	99.8	100	pl-b-i-12	99.7	100	pl-b-i-14	99.4	100
pl-b-i-8	99.5	100	pl-b-i-9	99.9	100	pl-r-i-10	99.4	100
pl-r-i-11	99.6	100	pl-r-i-12	99.5	100	pl-r-i-14	99.1	100
pl-r-i-8	99.0	100	pl-r-i-9	99.5	100	tm-b-i-10	99.7	100
tm-b-i-11	99.5	100	tm-b-i-12	99.6	100	tm-b-i-14	99.6	100
tm-b-i-8	99.5	100	tm-b-i-9	98.9	100	tm-r-i-10	99.2	99.8
tm-r-i-11	99.6	99.8	tm-r-i-12	99.4	99.8	tm-r-i-8	97.9	99.6
tm-r-i-9	99.5	99.7	tm-r-r-8	99.9	100	zc-r-r-10	83.9	99.4
zc-r-r-11	91.0	99.3	zc-r-r-12	91.6	99.3	zc-r-r-14	93.6	99.8
zc-r-r-8	84.7	98.8	zc-r-r-9	85.9	99.3	cr-r-r-8	**100**	99.9
lb-r-r14	**100**	99.7	tm-r-r-8	**100**	99.8			
						118 autres fontes	100	100

Tableau 8.5 : Précision de la segmentation en mots avec (a) ScanWorX et (b) notre seuillage contextuel.

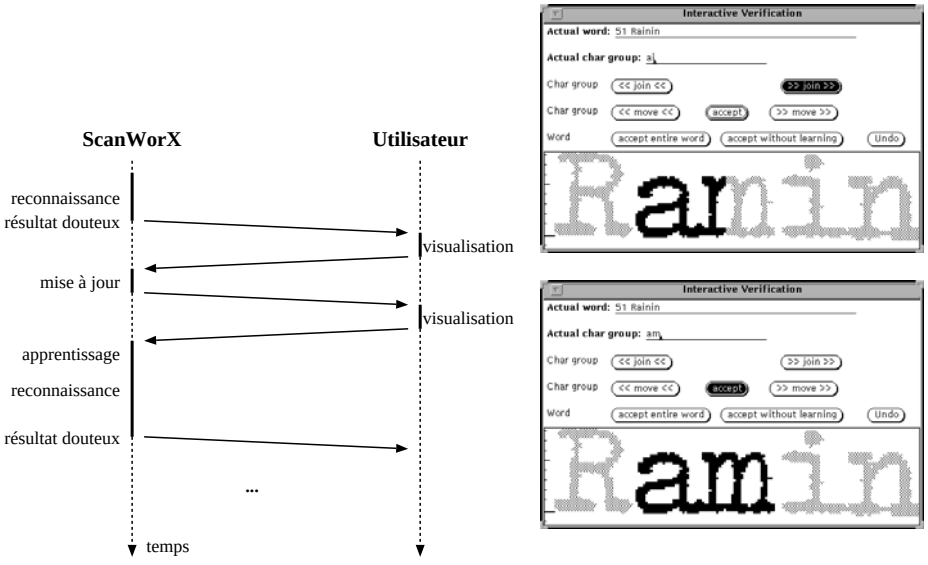


Figure 8.6 : Mode d'apprentissage interactif du logiciel ScanWorX.

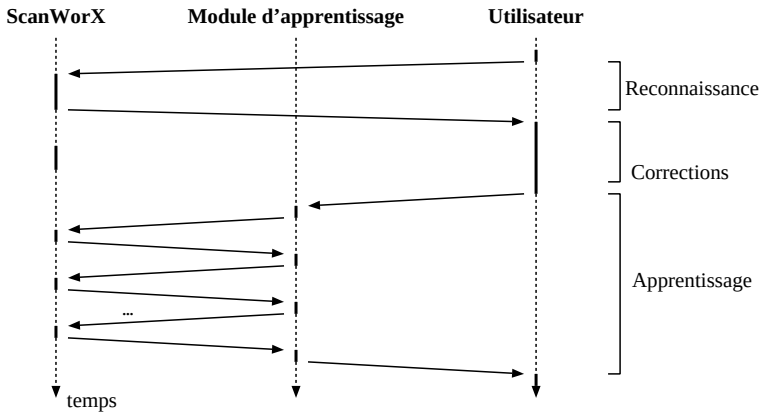


Figure 8.7 : Idée du module d'apprentissage.

Fonte	%	Fonte	%	Fonte	%	Fonte	%
ag-b-r-12	88	cr-b-i-10	70	cr-b-i-12	50	cr-b-i-14	50
cr-b-r-11	17	cr-b-r-12	33	cr-b-r-14	33	cr-b-r-8	73
cr-r-i-14	67	cr-r-r-11	50	cr-r-r-14	11	cr-r-r-8	59
hv-b-r-12	89	hv-b-r-14	75	hv-r-r-10	83	lb-b-r-14	86
lb-r-i-10	89	lb-r-r-14	33	lb-r-r-9	80	nc-b-r-10	57
nc-b-r-11	88	pl-b-i-14	29	pl-b-r-9	88	pl-r-i-10	62
pl-r-i-11	27	pl-r-i-12	15	pl-r-i-14	16	tm-b-i-14	62
tm-r-i-12	80	tm-r-i-14	64	tm-r-r-10	77	tm-r-r-12	75
						142 autres fontes	>90

Tableau 8.6 : Transactions d'apprentissage correctement transmises à ScanWorX.

fonte et les métriques correspondantes, on peut positionner assez facilement l'enveloppe de chaque caractère.

Notre algorithme se base sur les enveloppes rectangulaires pour isoler les résultats erronés. Il tente de faire correspondre un élément atomique fourni par l'OCR avec une suite de caractères corrects, en se basant sur les enveloppes rectangulaires. Tant que la frontière droite n'est pas commune, on fusionne avec le prochain atome, ou on englobe le caractère suivant. Notre mise en oeuvre risque de mal interpréter des taches qui chevaucheraient l'enveloppe des caractères.

8.5.2 Expériences

L'efficacité de notre module se mesure par sa capacité à résoudre toutes les situations douteuses détectées par ScanWorX. Il se peut en effet qu'une transaction d'apprentissage soit abandonnée. Parfois, notre analyseur ne parvient pas à trouver la correspondance dans le texte correct. Nous devons aussi subir une limitation de ScanWorX, qui interdit d'apprendre un groupe de plus de cinq lettres.

Nous avons appliqué notre pilotage automatique de l'apprentissage ScanWorX sur les 174 documents LaTeX scannés de notre base de données. Le tableau 8.6 présente les proportions de situations douteuses correctement détectées par notre module. Globalement, nous estimons que l'apprentissage est bien piloté, d'autant que le nombre de situations douteuses varie énormément d'une fonte à l'autre. Par exemple, pour la fonte `zc-r-r-10`, notre méthode ne gaspille que huit occasions d'apprentissage, sur plus de 1000, alors qu'il y a six occasions en tout pour la fonte `cr-b-r-12`.

Quant à l'amélioration des performances due à l'apprentissage, elle est propre au moteur de reconnaissance de ScanWorX, dans lequel nous ne sommes nullement impliqués. Comme cette question des gains apportés par l'apprentissage est toutefois des plus intéressantes, nous avons tout de même mesuré la précision de ScanWorX, une fois sans apprentissage, et une autre fois en utilisant les connaissances apprises via notre module. Le tableau 8.7 rapporte les résultats où la précision sans apprentissage était inférieure à 99.3%. Nous indiquons aussi les trois cas où l'apprentissage était légèrement défavorable. Globalement, l'apprentissage permet d'effectuer un bond dans la précision, p. ex. de 83 à 99% pour la fonte `zc-r-r-12`.

Notons que l'apprentissage offert par ScanWorX ne porte pas sur la segmentation en mots. Toutefois et

Fonte	(a)	(b)	Fonte	(a)	(b)	Fonte	(a)	(b)
ag-b-i-10	97.9	99.9	ag-b-i-11	98.4	99.8	ag-b-i-12	98.5	99.7
ag-b-i-14	97.6	99.9	ag-b-i-8	99.2	99.7	ag-b-i-9	98.9	99.6
ag-r-i-10	98.6	99.8	ag-r-i-14	98.0	99.4	ag-r-i-8	98.4	99.4
ag-r-i-9	98.8	99.4	ag-r-r-14	99.2	99.8	ag-r-r-9	99.2	99.5
cr-b-i-9	99.3	99.8	cr-r-i-14	99.6	**99.4**	hv-r-i-11	97.9	99.9
hv-r-i-12	99.0	99.9	hv-r-i-14	96.9	99.8	pl-b-i-9	99.1	99.7
pl-r-i-10	98.9	99.6	pl-r-i-8	97.5	98.5	pl-r-i-9	98.7	**97.6**
tm-b-i-12	99.7	**99.6**	tm-b-i-8	99.0	99.7	tm-b-i-9	98.9	99.4
tm-r-i-10	98.2	98.2	tm-r-i-11	98.5	99.6	tm-r-i-8	94.2	95.3
tm-r-i-9	96.4	98.0	tm-r-r-8	99.3	99.8	zc-r-r-10	80.9	98.1
zc-r-r-11	86.4	98.0	zc-r-r-12	83.8	99.0	zc-r-r-14	87.2	99.1
zc-r-r-8	80.1	95.8	zc-r-r-9	79.9	98.2			

Tableau 8.7 : Précision de ScanWorX (a) avant et (b) après apprentissage sur le document analysé.

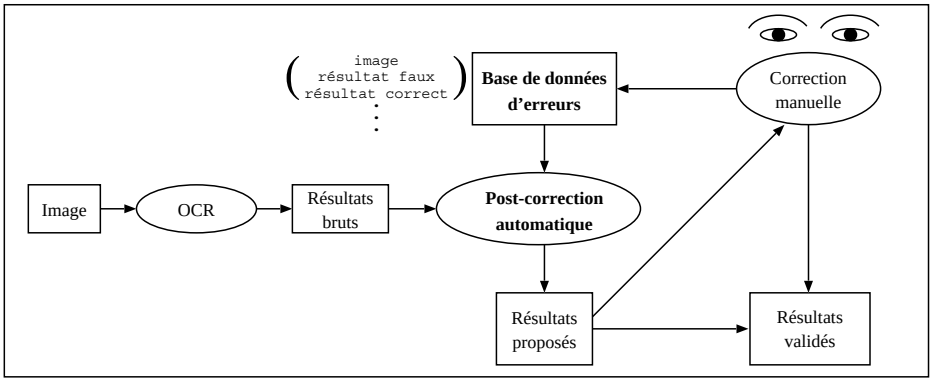


Figure 8.8 : Idée de la méthode de post-correction automatique.

pour autant que le texte respecte la langue prévue, la précision de ScanWorX sur la segmentation en mots augmente s'il fait moins d'erreurs sur la reconnaissance des caractères. En effet, les connaissances lexicales peuvent aider à détecter les espaces seulement si les caractères sont correctement reconnus.

D'autres indications sur la qualité de l'apprentissage de ScanWorX transparaissent dans la section 9.4.3, où nous montrons comment l'apprentissage d'un bloc de texte peut améliorer la précision sur un autre bloc.

8.6 Post-correction d'OCR

8.6.1 Méthodologie

Le côté frustrant des logiciels d'OCR réside surtout dans la prolifération des erreurs systématiques. L'apprentissage sert à ajuster les paramètres internes (seuils, poids, etc.) de la méthode d'analyse, afin de forcer sa décision dans une situation donnée. Suivant les techniques utilisées (p. ex. réseaux de neurones), les effets de ces modifications sont mal maîtrisés, et peuvent faire surgir de nouveaux types d'erreurs répétitives. Nous proposons ici une approche alternative, et plus générale, schématisée sur la figure 8.8 : elle consiste à garder l'analyseur tel quel, et à procéder à un post-traitement des résultats pour détecter une erreur déjà rencontrée.

Le module de post-correction automatique est un véritable outil de reconnaissance, avec toutefois deux particularités. Premièrement, l'analyse ne porte pas uniquement sur l'image mais aussi sur l'interprétation préalable de l'OCR. Deuxièmement, les éléments à reconnaître forment un ensemble d'exceptions, généralement assez restreint.

Le principe de notre analyseur revient à stocker les corrections dans une base de connaissances, afin d'en détecter de nouvelles occurrences dans les futurs résultats bruts. Un avantage de cette approche tient aux faibles besoins en calcul, puisqu'on peut filtrer les portions de résultats intéressants par une simple comparaison des valeurs qui caractérisent la situation erronée. Les critères possibles comprennent les codes ASCII proposés par l'OCR (voire la confiance associée), les dimensions des enveloppes, et la fonte. La mise en correspondance des images ne s'applique finalement que sur une petite proportion des résultats bruts. De

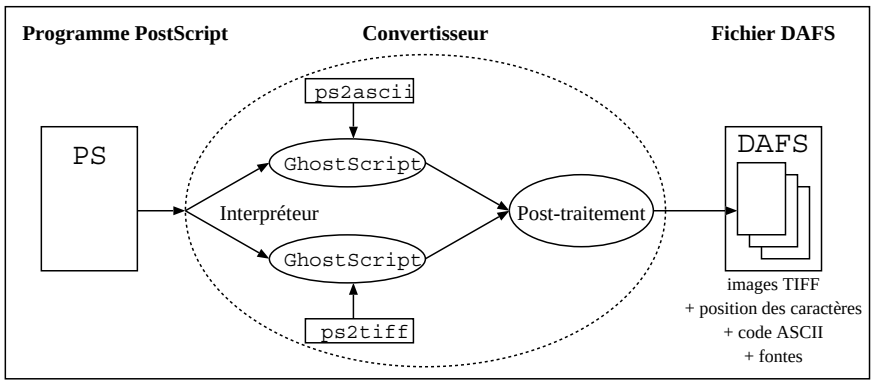


Figure 8.9 : Idée du convertisseur PostScript.

plus, on peut se contenter d'une mesure de dissemblance triviale (distance de Hamming) et d'un seuil assez bas, car chaque correction reste très spécialisée.

8.6.2 Expériences

Une première version de l'outil de post-correction automatique d'OCR a été testée comme extension du logiciel ScanWorX [29]. Les situations où ScanWorX commet beaucoup d'erreurs sont naturellement les plus intéressantes pour l'évaluation. Nous rapportons ici une expérience portant sur un échantillon de notre base de données, formatté avec la fonte `zc-r-r-12` (document LaTeX, scanné, sans dégradation).

Pour cette expérience, les erreurs de segmentation ont été filtrées, car notre méthode ne porte que sur la reconnaissance de caractères. Sur le document testé, ScanWorX atteint une précision de 79.9 % caractères reconnus. Cette performance extrêmement médiocre s'explique par les aspects non standards de la fonte Zapf-Chancery.

Nous avons d'abord lancé notre outil de post-correction en mode d'apprentissage, sur la base de la version correctement reconnue. Le nombre d'erreurs détectées entre les deux versions se montait à 634. Puis nous avons appliqué notre algorithme de post-correction sur les résultats bruts fournis par ScanWorX. Sur les 277 catégories d'erreurs stockées dans la base de connaissance, 71 ont pu être appliquées sur d'autres occurrences que celle qui avait été apprise. Après la post-correction, les résultats atteignait une précision de 99.46%. Les erreurs résiduelles correspondent aux cas où une post-correction a malencontreusement été réappliquée sur des résultats qui étaient justes. Notons encore que les cinq catégories d'erreurs les plus fréquentes concernaient en tout 143 occurrences, ce qui illustre le caractère répétitif des erreurs dans cette expérience.

Nous avons aussi cherché à appliquer notre technique *sans mettre en correspondance les images*. Dans ce cas, la post-correction n'utilise comme critère que le texte proposé et les enveloppes rectangulaires des caractères. Notre outil de post-correction introduit naturellement plus d'erreurs, mais la précision est tout de même passée de 79.9% à 94.96%.

En résumé, cette petite expérience nous apprend une chose importante. Lorsque, dans une application, les données à analyser sont très mal reconnues par l'OCR dont on dispose, et même si ce dernier n'est pas adaptatif, il est possible d'écrire une extension qui offre la fonctionnalité d'apprentissage incrémentale. Comme on peut se contenter d'un algorithme simple, dont le codage est très facile, cette approche est peu coûteuse. Nous avons constaté que la précision pouvait augmenter jusqu'à un niveau satisfaisant. On signalera encore qu'il serait intéressant de faire le lien entre notre méthode et d'autres travaux portant sur l'analyse des résultats d'OCR [35, 66].

8.7 Convertisseur PostScript

8.7.1 Méthodologie

La motivation de notre convertisseur PostScript provient des applications de restructuration de documents électroniques (cf. section 1.2.1). L'idée maîtresse consiste à passer par l'image, pour s'en remettre ensuite aux techniques de reconnaissance de documents (p. ex. segmentation). Toutefois, les images de pages ne sont pas les seules données à extraire d'un document PostScript. En effet, on doit pouvoir s'affranchir de

l'OCR, puisque les caractères affichés correspondent finalement à une opération explicite dans le programme PostScript. De plus, ce programme nous indiquera quelle est la police courante au moment du dessin de chaque caractère, de même que la position, la largeur et la hauteur du signe, voire la couleur utilisée. Par contre, l'odre de lecture n'est pas forcément préservé.

Comme schématisé sur la figure 8.9, notre convertisseur utilise un interpréteur PostScript, et génère un document DAFS (cf. section 4.3.3), qui contiendra des informations sur la segmentation (page, signes), le texte et la fonte. Ces résultats constituent une base pour les analyses ultérieures, par exemple la reconstruction des mots, des lignes et des paragraphes (puisque ces informations ne sont pas forcément véhiculées en PostScript).

La mise en oeuvre du convertisseur se base sur un script `ps2ascii`, dont une version est distribuée avec l'interpréteur `ghostscript`. Certaines retouches de ce script PostScript ont été nécessaires, pour obtenir le détail de chaque caractère et pour extraire le maximum d'indications sur la fonte. A signaler que plusieurs utilitaires similaires sont disponibles, comme `ps2html` ou `pstotext`. Ceux-ci utilisent des heuristiques pour détecter les sauts de lignes ou les titres. Le prototype de Benoît Poirier (cf. section 2.1.5) est probablement le plus avancé dans le domaine porteur que représente la reconnaissance assistée de documents PostScript.

8.7.2 Expériences

Etant donné que notre analyseur se borne à traduire les informations véhiculées par le document PostScript, sans aucune tentative d'interprétation, la question de l'évaluation ne porte pas sur la précision, mais plutôt sur la robustesse.

En effet, la principale difficulté provient de la variété des documents PostScript, et de l'absence de conventions universelles. Par exemple, PostScript prévoit une manière conventionnelle de gérer les fontes. Suivant la provenance du fichier (par exemple les documents produits avec LaTeX), ces conventions sont parfois violées.

Notre analyseur a été testé avec succès sur des documents provenant de sources variées, comme LaTeX, Netscape, MsWord ou FrameMaker. Seule la détection du nom de la fonte n'est pas toujours possible.

Notons que la fonte pourrait facilement être découverte par notre outil de reconnaissance a posteriori (cf. section 8.3). En effet, notre convertisseur PostScript indique la position de chaque caractère. Il suffirait de faire correspondre un des caractères du document avec un représentant de toutes les fontes possibles. L'absence de bruit renforce les chances de succès. Cette approche permettrait de transformer le résultat de *discrimination* des fontes en un résultat de *compréhension* des fontes (cf. section 7.3.1).

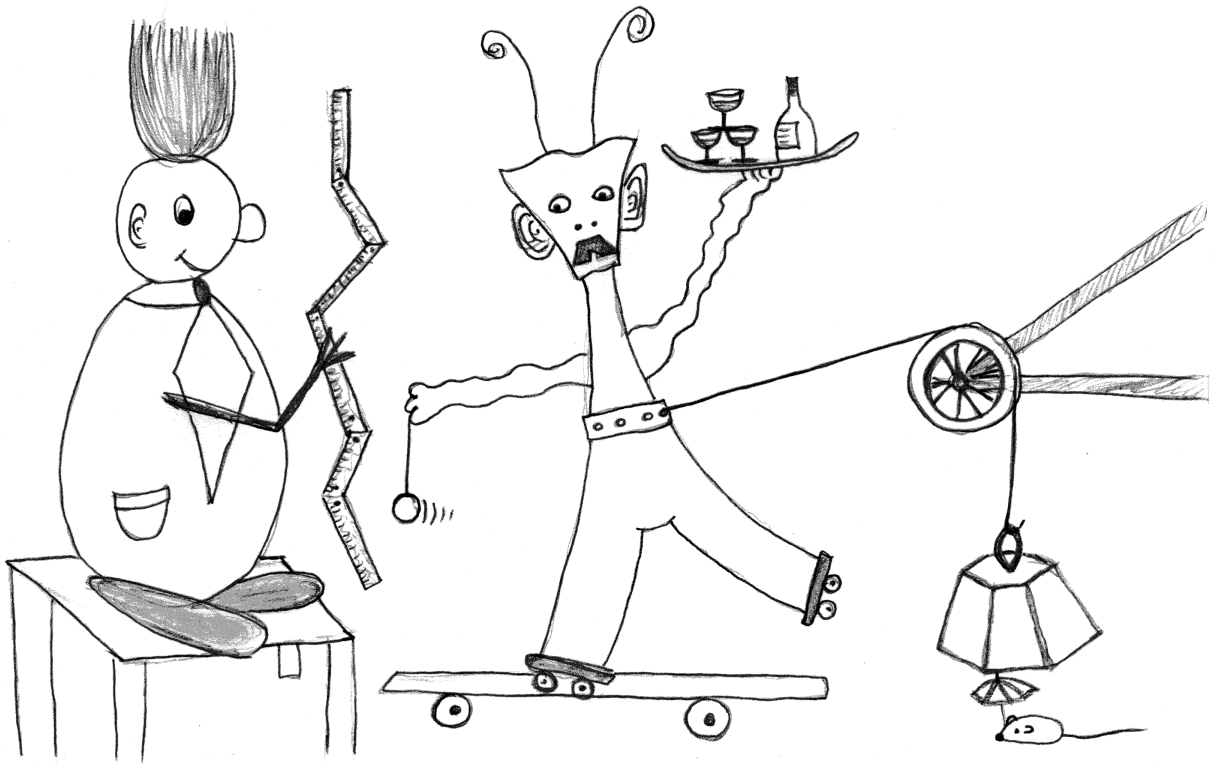
Conclusion

L'approche assistée remet en cause le cadre usuel qui préside au développement d'analyseurs. Nous avons esquissé une nouvelle voie qui vise à mieux valoriser les techniques d'analyse existantes. Les expériences nous ont montré que nos analyseurs, malgré leur simplicité, affichent parfois des performances étonnantes.

Un élément-clé de la réalisation d'analyseurs réside dans la démarche d'évaluation des performances. Le chapitre suivant discute une extension aux modèles de coûts classiques, afin de tenir compte de l'adaptabilité des outils d'analyse.

Chapitre 9

Modèles d'évaluation en reconnaissance assistée



Après avoir proposé des solutions pour le *développement* de systèmes de reconnaissance assistée, nous allons maintenant aborder un autre volet de l'analyse d'images de documents, avec la question de l'*évaluation des performances*.

Dans le cas d'outils de reconnaissance entièrement automatiques, la communauté scientifique prend cette question très au sérieux [157, 187, 147, 142], et les méthodes d'estimation utilisées nous apparaissent tout à fait satisfaisantes. En gros, elles consistent à appliquer l'analyseur sur une collection de données, puis à calculer une mesure qui renseigne sur la proportion de résultats erronés.

En revanche, nous estimons que cette démarche classique s'applique mal à la reconnaissance assistée, car elle occulte totalement le rôle actif de l'opérateur humain. Le problème s'est révélé plus complexe que prévu. Par exemple, on aimerait bien mesurer si le système est capable de s'améliorer durant une session de reconnaissance. De même, il serait intéressant de comparer plusieurs manières de piloter la reconnaissance d'un même document. On peut aussi se demander dans quelles circonstances le temps passé à adapter les analyseurs est compensé par les gains dus à l'amélioration des performances.

Ce chapitre propose un nouveau cadre pour l'évaluation des coûts occasionnés par des systèmes de reconnais-

sance interactifs et adaptatifs. La section 9.1 définit la notion de coût en faisant référence aux interventions de l'utilisateur. La section 9.2 établit les bases formelles d'une modélisation des coûts de reconnaissance, à l'aide de nouvelles notations. Le modèle théorique est ensuite utilisé dans deux contextes pratiques. La section 9.3 montre comment on peut simuler une partie du modèle de coûts, afin de mieux comprendre le comportement des systèmes de reconnaissance. La section 9.4 présente une série d'expériences avec des analyseurs existants, et illustre les avantages de l'approche assistée sur l'approche automatique.

9.1 Facteurs de coûts

Il est raisonnable et courant d'évaluer un système de lecture optique en mesurant les coûts occasionnés par le processus de reconnaissance. Une modélisation pertinente de la notion de coûts n'est en soi pas évidente [147]. Avec son concept de reconnaissance assistée, le projet *CIDRE* introduit une difficulté supplémentaire, dans la mesure où le comportement des analyseurs automatiques n'est plus indépendant des interventions de l'opérateur humain.

9.1.1 Coûts de reconnaissance et approches classiques

Plusieurs aspects du processus de reconnaissance sont susceptibles de constituer une source de coûts, notamment :

(a) l'acquisition et la maintenance de l'équipement matériel et logiciel nécessaire;

(b) l'utilisation des ressources informatiques (temps CPU, place mémoire, occupation du scanner, etc.);

(c) la main d'oeuvre sollicitée durant la phase d'exploitation, lorsque le système est utilisé de manière opérationnelle;

(d) les conséquences liées à l'utilisation d'une donnée erronée, par exemple dans une application de lecture de chèques.

Les points (a) et (d) se prêtent davantage à une analyse économique dans un contexte donné, qu'à une modélisation générale. Le point (b) se réduit essentiellement à la notion de complexité algorithmique. Il s'agit d'un problème séparé et bien défini, et on se bornera ici à rappeler que l'optimisation des analyseurs est souvent imposée par des contraintes de temps réel. C'est donc au point (c) que se rattachent les modèles de coûts. En effet, on cherche à estimer le temps qu'un opérateur humain doit consacrer pour superviser la reconnaissance jusqu'à l'obtention de résultats fiables.

L'approche traditionnelle se base sur le schéma d'exploitation suivant : le système d'analyse génère automatiquement une solution, qu'un opérateur doit ensuite corriger manuellement. Les coûts sont supposés proportionnels aux erreurs à corriger, et on utilise comme métrique une *distance d'édition* entre la solution produite par le système et le document correctement structuré. On évalue un outil de reconnaissance en l'appliquant sur une collection de test et en mesurant les taux d'erreur moyens. Les modèles classiques ont permis de découvrir des résultats pratiques très intéressants. Chow [46] définit par exemple une notion de *taux de rejet optimal*, en utilisant une mesure relative du coût de rejet par rapport au coût d'erreur.

Même sous l'angle classique, l'applicabilité d'un modèle de coût n'est pas exempte de critiques, car il est difficile de s'approcher des conditions réelles :

- L'interface graphique influence directement les temps de correction. Suivant les applications, elle peut offrir des fonctions plus ou moins riches pour faciliter le travail.
- Il n'est pas réaliste d'associer un coût aux opérations ponctuelles, car le contexte joue aussi un rôle : corriger deux erreurs dans le même mot coûte moins que dans des lignes séparées.
- Il serait plus précis de distinguer plusieurs étapes dans une correction, telles que détection de l'erreur, identification de la solution correcte, et édition.
- Le concept d'édition de chaînes convient bien à l'OCR, mais n'est pas forcément naturel pour les autres types de résultats comme la segmentation ou la reconnaissance de fontes.
- Définir des paramètres théoriques est d'intérêt mineur par rapport aux expériences en vraie grandeur chez l'utilisateur final.

Ce dernier point mérite une attention particulière. Nous sommes convaincus que la meilleure manière d'évaluer un logiciel de reconnaissance consiste à mesurer le temps passé par les opérateurs humains en phase

d'exploitation. Par exemple, il aurait été très enrichissant de tenter une comparaison entre des prototypes implémentant les trois styles de dialogues présentés dans la section 3.4. La démarche consisterait à préparer (i) des versions opérationnelles de prototypes, (ii) un ensemble de documents à reconnaître, et (iii) une équipe d'opérateurs volontaires. Le système de reconnaissance devrait générer une trace de tous les traitements effectués, et des interventions de l'utilisateur. Une analyse détaillée de ces traces apporterait des arguments incontestables pour évaluer la qualité des prototypes. Malheureusement, ce genre d'expériences est souvent hors de portée des équipes de recherche, bien que l'intérêt scientifique nous semble très élevé.

A défaut de pouvoir nous lancer dans des expériences dans le terrain, nous avons cherché à étendre les modèles de coûts pour y introduire l'influence de l'utilisateur.

9.1.2 Influence de l'utilisateur

Un environnement de reconnaissance de documents assistée doit intégrer l'analyse automatique et l'édition manuelle dans un seul système capable d'apprentissage. Dans un environnement adaptatif, le taux d'erreur moyen n'est plus suffisant pour juger de la qualité d'un analyseur, car cette mesure ne rend pas compte des possibilités d'amélioration au cours du temps. Une première approche consiste à analyser ces deux critères séparément.

Il y a encore deux aspects qui doivent être pris en compte. D'une part, l'apprentissage n'est pas forcément gratuit : il ne faut pas négliger le temps passé à essayer d'améliorer les analyseurs. Les transactions d'apprentissage peuvent certes se déclencher de manière transparente (après chaque correction manuelle), mais lorsque ce n'est pas le cas, le scénario optimal est une affaire de compromis entre les gains liés à l'amélioration des performances, et le temps additionnel passé dans les interactions d'apprentissage. D'autre part, le taux moyen d'amélioration n'est pas une mesure très pertinente, car l'effet d'apprentissage est irrégulier. En fait, le coût global pour reconnaître un volume donné dépend de l'ordre dans lequel les éléments sont analysés, ainsi que de l'ordonnancement des différentes étapes d'analyse et d'apprentissage.

Il faut en conclure que la dynamique d'une session de reconnaissance exerce sur les coûts une influence qui mérite d'être étudiée.

On pourrait comparer une session de reconnaissance avec un itinéraire qui doit mener un engin (le système de reconnaissance) d'un point de départ (les données brutes) vers une destination (le document correctement structuré). Les mouvements de déplacement sont caractérisés par la vitesse (taux de reconnaissance) et l'accélération (taux d'amélioration). Les différentes trajectoires possibles correspondent aux différentes manières d'enchaîner les traitements. Suivant cette métaphore, l'approche classique estime la durée du trajet sur la base de la vitesse moyenne (voire maximale) de l'engin. Pour notre part, nous nous intéressons à l'écoulement du temps tout au long du parcours. Le but final serait de mettre en évidence des principes qui aident à manoeuvrer l'engin quelles que soient les conditions du terrain.

En résumé, les méthodes d'évaluation classiques s'évertuent à mesurer un *taux d'erreur moyen*; l'effort de modélisation porte sur les conditions d'utilisation, le comptage des erreurs, ou une adaptation des coûts selon le type d'erreurs. Nous nous attaquons à des problèmes sur lesquels bute l'approche classique. Par exemple, comment départager un OCR affichant une précision de 99%, et un autre outil moins précis au départ, mais doué d'apprentissage incrémental? Comment caractériser l'évolution des coûts à mesure que la session de travail progresse? Parmi tous les scénarios possibles pour accomplir une certaine tâche de reconnaissance avec des analyseurs donnés, lequel est de coût minimal?

9.2 Formalisation

Parvenus à ce stade de la discussion, nous devinons le phénomène essentiel qui échappe aux approches classiques : les résultats d'analyse dépendent de l'état du système au moment où l'analyse est commandée, et cet état dépend notamment des interventions de l'utilisateur. Toutefois, il nous manque les moyens d'expression adéquats pour mener le débat.

Cette section tente de formaliser la discussion, en introduisant des notations originales. Tout d'abord, nous fixons des conventions pour représenter un enchaînement d'opérations durant l'utilisation d'un système de reconnaissance assisté. Ensuite, nous nous inspirons des grammaires formelles pour distinguer trois stratégies d'analyse typiques, suivant le rôle attribué à l'utilisateur. Finalement, le modèle de coût proprement dit vient se greffer au-dessus de nos notations : des paramètres sont introduits pour le coût des opérations atomiques, et le modèle permet de formuler le coût total d'une session de reconnaissance.

Expression	Commentaire
$(P_1 P_2)$	séquence
$(P_1 \cup P_2)$	alternative
$(P_1)^*$	fermeture itérative
$p[A^c]$	OCR sur l'entité *spécifique* p
$Page[A^c]$	motif *générique* pour l'OCR sur une page
A^c	abbréviation pour $X[A^c], \forall X$
$e_1[A^c]e_2[M^f]$	séquence (d'abord l'OCR sur e_1)
$(e_1, e_2)[A^c M^f]$	abbréviation pour $e_1[A^c]e_1[M^f]e_2[A^c]e_2[M^f]$

Tableau 9.1 : Notations pour les motifs de session.

9.2.1 Modèles de session de reconnaissance

La situation que nous discutons se limite au problème de la reconnaissance de structure physique à partir d'images de documents. L'état courant d'une solution est organisé en une hiérarchie d'entités. Dans notre modèle, une entité e est un noeud d'arbre qui supporte trois interprétations correspondant aux tâches de bases :

- interprétation textuelle c, pour attacher à e une chaîne de caractères;
- interprétation typographique f, pour attacher à e une fonte;
- interprétation de segmentation s, pour attacher à e un sous-arbre et une enveloppe géométrique.

Chaque interprétation est encore qualifiée par un statut pour indiquer si ce résultat a été certifié, s'il est encore inconnu, où quel est le degré de confiance dont il jouit.

Nous appelons une *session* le scénario suivi lors d'une exécution du système. Une session se compose d'une séquence d'opérations de bases appelées *transactions*. Nous proposons un modèle simple avec trois types de transactions, appliqués à chaque champ d'interprétation :

- analyse automatique – $A \in \{A^c, A^f, A^s\}$;
  le système possède trois analyseurs, un pour chaque interprétation à calculer (texte, fonte, segmentation); leurs paramètres d'entrée sont l'entité à analyser, la base de connaissances à utiliser, et éventuellement d'autres options de configurations;
- édition manuelle – $M \in \{M^c, M^f, M^s\}$;
  l'opérateur humain peur modifier directement n'importe quelle partie de la solution courante (éditer une chaîne, choisir une fonte, segmenter à la main). Notons que le détail des actions dépendra de l'interface homme-machine;
- apprentissage incrémental – $L \in \{L^c, L^f, L^s\}$;
  les analyseurs offrent un moyen de mettre à jour leurs bases de connaissances, afin d'y intégrer une solution connue (c.-à-d. corrigée).

Ces neuf symboles forment l'alphabet de base sur lequel nous construisons notre notation. Un *motif de session* décrit une règle de syntaxe sur l'enchaînement des transactions. Le tableau 9.1 présente les notations que nous proposons pour exprimer des motifs. Le recours à une représentation inspirée par les langages réguliers permet d'encoder des contraintes sur les motifs. Par exemple, on peut exprimer un schéma d'analyse complet $Volume[A]$ par $(Page[A^s]\ (Page[A^f A^c] \cup Page[A^c A^f])\ )^*$.

9.2.2 Stratégies d'organisation d'une session

Maintenant que nous savons décrire des sessions de reconnaissance, nous sommes en mesure de discuter différentes manières d'organiser les opérations dans un système. Nous allons expliquer trois approches extrêmes, résumées dans le tableau 9.2.

Par lots (BATCH) : Le système n'offre pas d'apprentissage incrémental. Tout le calcul est groupé dans une première étape de la session, qui se poursuit par une phase d'édition où l'utilisateur corrige les résultats. Les motifs $\mathcal{P}_{batch}$ de l'approche par lots sont notés $Volume[AM]$. Cette approche reflète l'état de l'art tel qu'il se manifeste dans les systèmes commerciaux. Plus précisément, ces systèmes de reconnaissance sont certes encastrés dans des environnements interactifs, mais aucune forme d'apprentissage incrémental n'est

		BATCH	AMD	AUD
Utilisateur	rôle	passif	réactif	actif
	rétroactions	non	oui	oui
	expérience	inutile	inutile	utile
Motif	granularité	fixe	fixe	variable
	longueur	2 (constant)	n (connu)	m (inconnu)
	parcours	atomique	imposé	libre

Tableau 9.2 : Trois organisations de session extrêmes.

réellement intégrée. L'interactivité sert typiquement à corriger la segmentation avant d'appliquer l'OCR. Dans ce sens, les motifs correspondants s'exprimeraient par $(A \cup M)$. Parfois le système autorise une phase d'apprentissage initial où l'utilisateur doit produire manuellement la solution pour des échantillons typiques, et on aurait dans ce cas $((ML)^* A \cup M)$.

Assisté et dirigé par la machine (AMD) : Le système choisit la séquence d'entités et les analyseurs à appliquer. L'utilisateur est sollicité après chaque analyse pour corriger la solution, et éventuellement la soumettre à l'apprentissage incrémental. Les motifs $\mathcal{P}_{amd}$ sont notés $(A \cup (AM) \cup (AML))^*$. Au moins un projet de recherche adopte cette approche : Stabler [194] décrit un système spécialement développé pour la capture de gros volumes de données, qui permet une reconnaissance parfaite et semi-automatique dans une application de lecture de brevets.

Assisté et dirigé par l'utilisateur (AUD) : Le système est complètement sous le contrôle de l'utilisateur, qui peut appliquer la reconnaissance, l'apprentissage ou l'édition manuelle à l'endroit et au moment où il le désire. Les motifs $\mathcal{P}_{aud}$ sont notés : $(A \cup M \cup L)^*$. Notre projet *CIDRE* se positionne plutôt dans cette approche, dans le sens où l'utilisateur est libre d'interagir comme il l'entend. Par contre, nous prônons un mécanisme qui lance les analyses de manière transparente, c.-à-d. sans attendre une commande explicite. Notons que l'approche AUD est une généralisation des deux autres, car rien n'empêche l'opérateur d'agir de manière systématique.

9.2.3 Modèle de coût pour les interventions de l'utilisateur

Notre modèle simple sert à décrire les motifs de session. Nous allons maintenant le compléter en attribuant des coûts aux différentes opérations qui forment une session. Comme expliqué à la section 9.1.1, c'est une estimation du temps de travail de l'opérateur humain qui nous intéresse. Dans notre modèle, il y a trois types d'interventions manuelles possibles :

- Soumettre une entité à un analyseur dans une requête d'analyse. Nous supposons que cette opération est de coût constant u, quel que soit l'entité ou le type d'analyse. On pourrait par exemple imaginer que cette requête se déclenche à l'aide d'un menu contextuel associé à la représentation graphique de chaque entité.
- Soumettre une entité à un analyseur dans une requête d'apprentissage. Notre modèle y attache le même coût u, car le mécanisme d'appel devrait être similaire.
- Editer la solution courante. Les coûts seront différents selon la nature du résultat, car les mécanismes d'édition ne sont pas identiques. De plus, le temps d'édition va dépendre de la qualité des analyses automatiques. Il s'agit donc de modéliser les coûts d'édition de manière plus détaillée.

Modèle de coûts d'édition

Par *édition* il faut entendre la mise à jour manuelle de la solution courante, soit en corrigeant des résultats erronés dans une entité, soit en entrant les informations sans qu'il y ait eu analyse automatique. Le coût de correction dépend de la nature du résultat (taper des caractères, choisir une fonte, définir une enveloppe ou réarranger une découpe), ainsi que du statut de l'interprétation, car il est plus facile de détecter une erreur si ce résultat est marqué comme incertain.

Notre proposition de modèle de coût est résumée dans le tableau 9.3. Soit W_e^t l'ensemble d'erreurs laissées par l'analyseur $t \in \{c, f, s\}$ sur l'entité e, ou plus exactement l'ensemble des membres du sous-arbre nécessitant d'être encore édités pour aboutir à une interprétation correcte pour la tâche t. Nous faisons une distinction entre le coût δ_e^t d'une erreur *atomique*, et le coût Δ_e^t de correction de *tout un sous-arbre*. Pour l'OCR, nous

Symboles	
W_e^t	ensemble des e_i dans le sous-arbre de e possédant une valeur erronée pour la tâche $t \epsilon \{c, f, s\}$
δ_e^t	coût d'édition pour corriger l'erreur *atomique* de l'entité e pour la tâche t
Δ_e^t	coût d'édition pour corriger *toutes* les erreurs dans le sous-arbre de e pour la tâche t
Δ_e	coût d'édition total pour corriger toutes les erreurs dans le sous-arbre de e pour toutes les tâches
λ_e	coût de détection d'erreur, fonction du statut du résultat erroné e
φ_e	nombre minimal d'opérations d'insertion, d'effacement et de substitution pour corriger la segmentation de e
α, β, γ	coût d'édition atomique pour le texte, resp. la fonte et la segmentation
Relations	
(i)	$\delta_e^s = \lambda_e + \varphi_e \gamma$
(ii)	$\delta_e^f = \lambda_e + \beta$
(iii)	$\delta_w^c = \lambda_e + \alpha$
(iv)	$\Delta_e^t = \sum_{w \in W_e^t} \delta_w^t$
(v)	$\Delta_e = \sum_{t \in \{c,f,s\}} \Delta_e^t$

Tableau 9.3 : Notations pour le modèle de coût d'édition.

Propriété		Commentaire
$W^t_{e,<e>} = \emptyset$	(i)	Efficacité absolue
$W^t_{e,<ei,ej>} = W^t_{e,<ej,ei>}$	(ii)	Commutativité
$W^t_{e,<ei>} = W^t_{e,<ei,ei>}$	(iii)	Sémantique d'ensemble
$\lvert W^t_{e,<>}\rvert \geq \lvert W^t_{e,<ei>}\rvert$	(iv)	Effet uniquement positif
$W^t_{e,<e_1 \ldots e_n>} \subset W^t_{e,<e_1 \ldots e_{n-1}>}$	(v)	Stabilité

Tableau 9.4 : Propriétés de l'apprentissage, qui ne sont *pas* forcément garanties.

supposons qu'une correction atomique concerne uniquement le niveau mot. En revanche, nous prévoyons qu'une fonte peut être attribuée manuellement à n'importe quelle entité, au sens où tout le sous-arbre accepte la nouvelle valeur. La segmentation manuelle dépendra fortement de l'interface homme-machine, qui peut définir des commandes de haut niveau pour couper/coller, séparer/fusionner ou réarranger. Notre modèle adopte une approche naïve avec des insertions, suppressions, et substitutions de coût identique. Nous n'allons pas détailler l'influence du statut λ_e, mais le système devrait offrir des fonctionnalités de visualisation à plusieurs degrés (certain, douteux, inconnu) [91], et gérer l'adaptation des seuils [46].

Modèle de coûts de session

La définition d'un coût d'édition ne donne qu'une vue *statique* des interventions de l'utilisateur. Les aspects dynamiques sont essentiels, puisque nous voulons comparer diverses organisations des tâches dans des sessions entières. Le phénomène crucial, c'est que l'utilisation de l'apprentissage incrémental est censé réduire la quantité d'édition (il y aura moins d'erreurs à corriger), à un prix proportionnel au nombre de requêtes d'apprentissage. Un autre facteur réside dans le choix de la granularité de traitement, parce que l'analyse d'un bloc coûtera moins que l'analyse répétée de chacune des lignes qui le composent.

La première étape consiste à raffiner la définition de l'ensemble d'erreurs W_e^t afin de la paramétrer avec la séquence $e_1 \ldots e_n$ des échantillons appris. Ce paradigme peut alors servir à discuter des propriétés que la fonctionnalité d'apprentissage devrait idéalement garantir. Le tableau 9.4 établit certaines de ces propriétés, qui ne sont d'ailleurs pas respectées en pratique. La notation permet aussi de déterminer une fonction normalisée du potentiel d'apprentissage de e_1 sur e_2, par exemple :

$$\frac{(\lvert W^t_{e_2,<>}\rvert - \lvert W^t_{e_2,<e_1>}\rvert)}{(\lvert W^t_{e_2,<>}\rvert - \lvert W^t_{e_2,<e_2>}\rvert)} \tag{9.1}$$

Symboles	
$W^t_{e,<e_1 \ldots e_n>}$	ensemble d'erreurs lorsque l'analyseur t a appris e_1 jusqu'à e_n.
$C(o \S P)$	coût de l'operation o commandée après le motif P.
u	coût de lancement d'une requête.
$C^M(P)$	coût total des transactions d'édition.
$C(P)$	coût total du motif P.
Relations	
(i)	$\forall P, \forall e, C(e[A] \S P) = C(e[L] \S P) = u$
(ii)	$C(e[M^t] \S e_1[L^t] \ldots e_n[L^t] e[A^t]) = \sum_{e_i \in W^t_{e,<e1 \ldots en>}} \delta^t_e$
(iii)	$C(o_1 \ldots o_n) = \sum_{1 \leq i \leq n} C(o_i \S o_1 \ldots o_{i-1})$
(iv)	$C(P) = C^M(P) + C^A(P) + C^L(P)$

Tableau 9.5 : Notations pour le modèle de coûts des sessions.

Maintenant, nous pouvons faire le lien entre le coût d'édition, l'effet d'apprentissage et le nombre de requêtes, afin de définir le coût d'un motif complet de session (cf. tableau 9.5). Nous définissons le coût $C(o \S P)$ d'une transaction o en fonction de l'historique P de la session. Ce n'est pas le cas pour les requêtes aux analyseurs, dont le coût u est constant. Par contre, c'est de première importance pour l'édition manuelle, car l'effort de correction $C(e[M^t] \S P)$ dépend des erreurs $W^t_{e,<e_1 \ldots e_n>}$ laissées par l'analyseur, qui dépendent à leur tour de la connaissance accumulée jusqu'à ce moment. Bien sûr, il est aussi possible d'éditer sans analyse automatique préalable; on aura alors le même coût que lorsqu'aucun résultat correct n'est découvert.

Dans notre modèle, optimiser l'organisation pour un certain volume revient à trouver un motif P couvrant toutes les entités spécifiques, qui minimise le coût d'intervention total $C(P)$. Deux facteurs antagonistes prennent part à cette optimisation :

- le meilleur recours aux transactions d'apprentissage, afin de minimiser les erreurs à corriger;
- la longueur de P, puisque le coût augmente avec le nombre de transactions du type A et L.

Le coût d'un motif P_{batch} représente l'effort de correction $C(v[M] \S v[A])$ sur un volume v.

Le coût d'un motif de la famille P_{amd} contient une partie fixe xu dépendant de la granularité de traitement prédéfinie (p. ex. ligne par ligne), une partie variable yu proportionnelle aux nombre de corrections apprises, et le coût d'édition $C^M(P_{amd})$ influencé indirectement par le parcours (p. ex. l'ordre de lecture).

Pour l'approche AUD, l'utilisateur a la mission de trouver un motif $P_{opt} \in \mathcal{P}_{aud}$ tel que $C(P_{opt})$ est minimal.

9.3 Simulations

La formalisation proposée dans la section 9.2.1 définit la notion de coût indépendamment du comportement des analyseurs ou des particularités du document traité. Cette optique permet d'appliquer le modèle lors d'expériences dans des conditions effectives. Mais l'étude théorique peut encore être complétée par une modélisation de la qualité des résultats d'analyse. On aboutit alors à un modèle complet de système de reconnaissance, grâce auquel on peut procéder à toutes sortes de simulations. On a ainsi plus de liberté pour analyser l'influence des différents paramètres.

Nous tenons toutefois à émettre des réserves quant à la pertinence des hypothèses sous-jacentes, par exemple sur l'indépendance des erreurs. Nous avons voulu faire un premier pas vers la simulation d'un système de reconnaissance. Toutefois, le bien-fondé de notre modèle mériterait d'être argumenté par une analyse du comportement des analyseurs existants. Par ailleurs, une telle analyse permettrait d'estimer les paramètres du modèle d'une manière plus réaliste.

9.3.1 Modélisation des erreurs de reconnaissance

Pour reprendre les notations de la section 9.2.1, nous cherchons maintenant à exprimer l'ensemble d'erreurs $W^t_{e,<e_1 \ldots e_n>}$ de manière plus rigoureuse, comme le résultat d'une certaine fonction. Par souci de clarté, nous limiterons la discussion au cas de la reconnaissance de texte. Le tableau 9.6 résume les paramètres de notre modèle, dont voici les hypothèses :

- *Document :* un document est représenté par une suite de lignes $l_1 \ldots l_N$. La ligne l_i contient W_i mots.

N	nombre de lignes dans un document	100
W_i	nombre fixe de mots dans chaque ligne	6
K	nombre de catégories d'erreurs possibles	100
F_i	probabilité qu'un mot contienne l'erreur i	$\frac{1}{5i}$
U_i	probabilité d'assimiler l'erreur i lors d'un apprentissage	1
β	fraction du coût d'édition lors d'un apprentissage massif	0.1
u	coût de lancement d'une requête	0

Tableau 9.6 : Paramètres du modèle de simulation, et valeurs choisies.

- *Erreurs :* il y a un nombre prédéfini K de catégories d'erreurs possibles. L'une de ces erreurs serait typiquement de confondre 'e' et 'c'. Dans une phase initiale, les erreurs potentielles sont distribuées aléatoirement sur tout le document, simulant ainsi les résultats d'un analyseur non entraîné. L'erreur i ($1 \leq i \leq K$) apparaît dans un mot avec une probabilité F_i.
- *Apprentissage :* l'unité d'apprentissage est le mot. Lorsque l'analyseur a assimilé une erreur, celle-ci n'apparaîtra plus dans les analyses suivantes. Lors de l'apprentissage d'un mot, l'analyseur assimile une occurrence de l'erreur i avec une probabilité U_i.
- *Coûts :* l'unité de coût équivaut à l'édition d'un mot, quel que soit le nombre d'erreurs qu'on doit y corriger. Notre modèle prévoit un mode spécial, où le coût d'édition s'abaisse à une valeur $\beta \leq 1$. Ce mode correspond aux cas où l'utilisateur retaperait lui-même tout le texte d'une partie du document. Nous pensons ici aux systèmes qui autorisent un apprentissage massif dans une phase initiale de reconfiguration.
- *Scénario :* le déroulement d'une reconnaissance est déterminé par un motif de session. Une grande variété de scénarios peuvent ainsi être simulés.

Il convient de relever les points les plus discutables de cette modélisation. Nous supposons que les erreurs sont distribuées aléatoirement, sans donner d'argument qui validerait l'hypothèse de l'indépendance entre erreurs. Nous introduisons les probabilités U_i comme un moyen d'atténuer la perfection de l'apprentissage, mais notre démarche n'a pas vraiment de fondement. Cet artifice ne rend en tout cas pas compte du phénomène où l'apprentissage peut engendrer de nouvelles erreurs.

9.3.2 Observations

Maintenant que nous disposons d'un modèle consistant de système de reconnaissance, il serait possible de procéder à d'innombrables simulations, en faisant varier n'importe quel paramètre. Pour notre part, nous visons les objectifs suivants :

- illustrer l'apport décisif de l'apprentissage incrémental, qui forme la base des approches assistées;
- montrer la variabilité des résultats selon les documents et les motifs, ce qui encouragerait les approches non dirigistes, plus aptes à adapter le schéma d'une session;
- expliquer de manière intuitive les différents schémas qu'on peut dresser avec des résultats observés.

Dans cette série de simulations, les paramètres sont fixés aux valeurs du tableau 9.6. Ainsi l'apprentissage est parfait ($U_i = 1, \forall i$), la plupart des erreurs sont peu fréquentes ($F_i = 1/5i$), et le lancement des requêtes est de coût nul ($u = 0$). La figure 9.1 illustre l'évolution des coûts à mesure que la session avance, pour les trois scénarios suivants : une version P_{batch} de base, une version P_{amd} où chaque correction est systématiquement apprise, et une version P_{batch+} où le motif contient un préfixe d'apprentissage massif sur les 10 premières lignes du document. La figure présente les résultats pour trois documents différents, c.-à-d. trois distributions aléatoires différentes des erreurs.

L'approche BATCH est ici évidemment pénalisée, puisque le recours à l'apprentissage est gratuit et sans effet néfaste. On constate par contre que les scénarios P_{amd} et P_{batch+} sont difficiles à départager; pour tirer des conclusions, il faut multiplier les échantillons. Le cas moyen, illustré sur la figure 9.2, montre bien l'avantage à long terme de l'adaptabilité. Dans cette figure, la pente de la courbe P_{batch} est constante et symbolise le taux d'erreur d'un OCR brut du marché. La pente de la première partie de la courbe brisée P_{batch+} représente le coût d'une acquisition entièrement manuelle, et il n'est en fait pas déraisonnable que ce soit moins coûteux qu'une solution automatique mal configurée. Plus on allonge la phase d'apprentissage massif de P_{batch+}, plus on va diminuer le taux d'erreur pour la fin du scénario. Par contre, la pente de cette deuxième partie de la

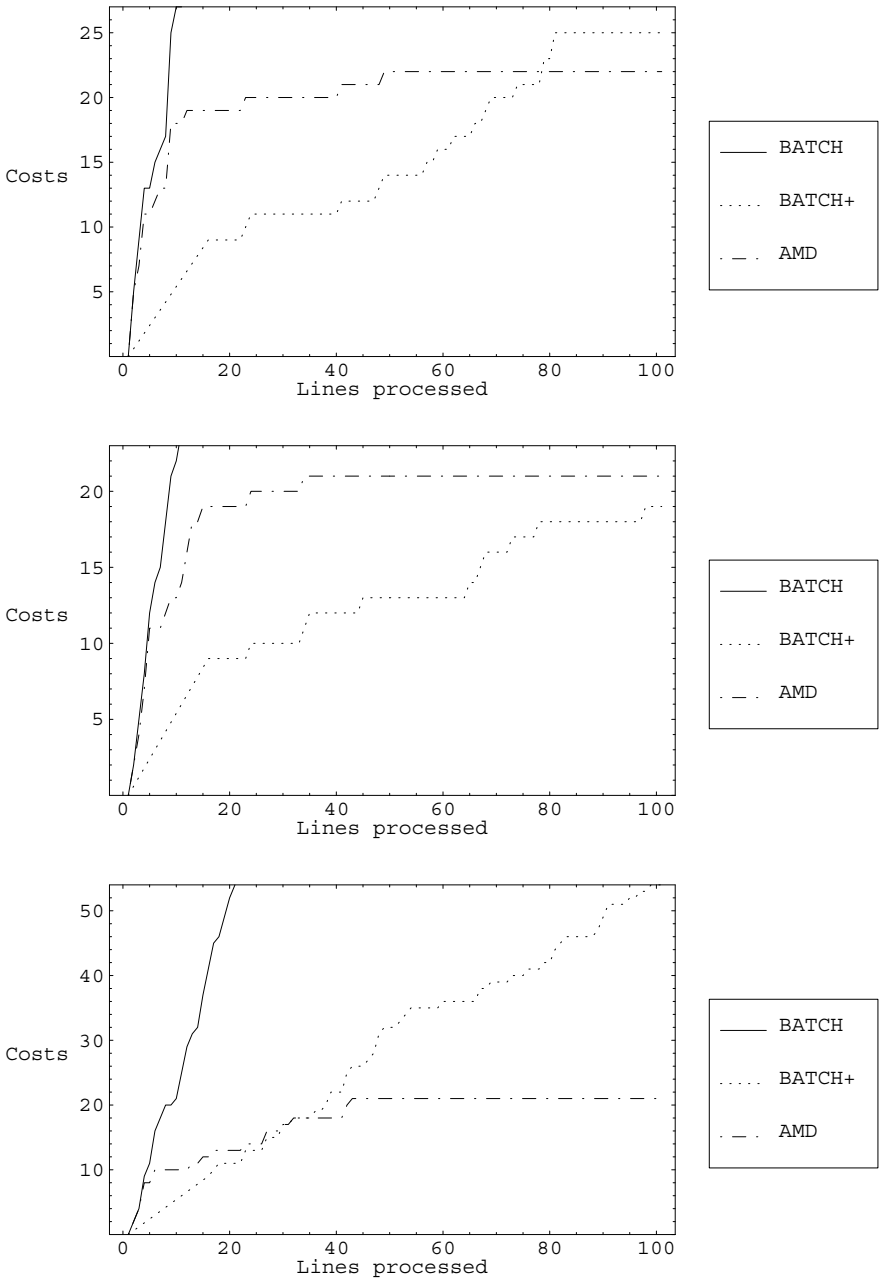


Figure 9.1 : Evolution des coûts pour 3 documents.

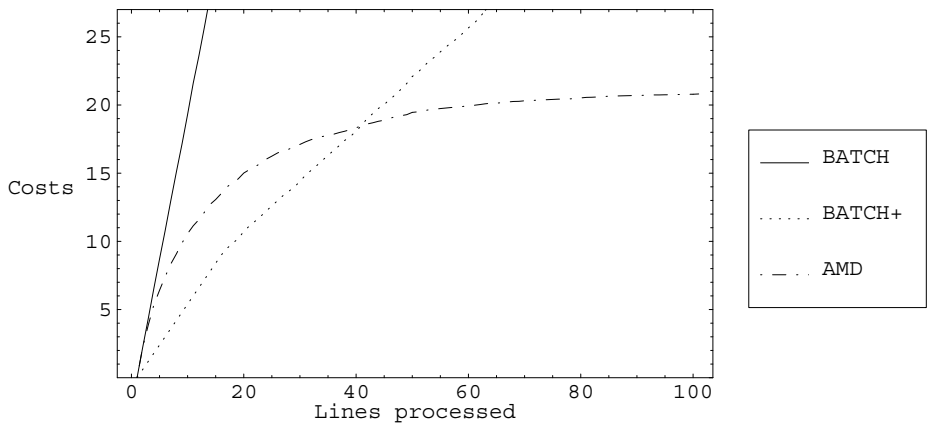


Figure 9.2 : Cas moyen d'évolution des coûts.

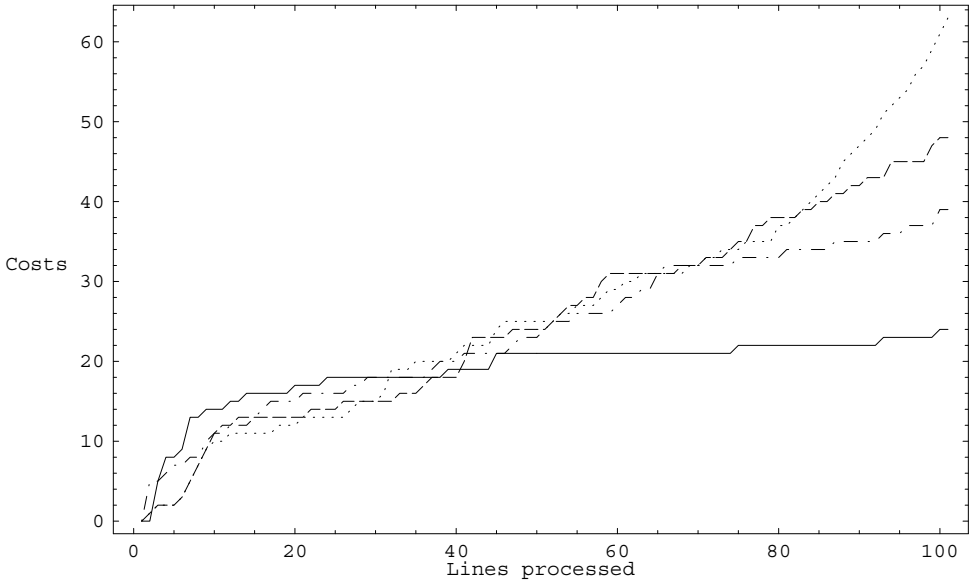


Figure 9.3 : Influence de l'ordre de traitement sur un scénario AUD.

courbe restera constante, puisque les erreurs non apprises continueront à apparaître avec une fréquence fixe. Seul le recours à l'apprentissage incrémental permet de tendre asymptotiquement vers un taux d'erreur nul.

La deuxième série de simulations met en jeu différentes manières de recourir à l'apprentissage incrémental. On va s'intéresser ici aux scénarios assistés où le quart des mots corrigés est soumis à l'apprentissage. Dans la figure 9.3, c'est toujours le même document qui est analysé, et le premier quart des corrections qui sont apprises, mais on a fait varier l'ordre de traitement des lignes. Dans la figure 9.4, un document est parcouru selon un ordre unique, mais le sous-ensemble de corrections apprises varie.

Au vu des décalages de coûts entre scénarios, qui passent parfois du simple au triple, deux remarques s'imposent :

- un modèle de coût qui suppose un scénario unique passe à côté d'un facteur de variation important;
- seul un schéma de reconnaissance souple permet de rationaliser le traitement au cas par cas.

C'est pourquoi l'approche *CIDRE* cherche à donner le plus de liberté à l'utilisateur, sans fixer à l'avance un scénario inflexible. De plus en pratique, le comportement de l'opérateur ne sera pas le fruit du hasard, car c'est le bon sens et l'expérience qui va le guider dans ses choix. On espère ainsi revaloriser ce travail manuel avec un léger aspect ludique : comment diriger les opérations pour terminer la reconnaissance dans un temps minimal.

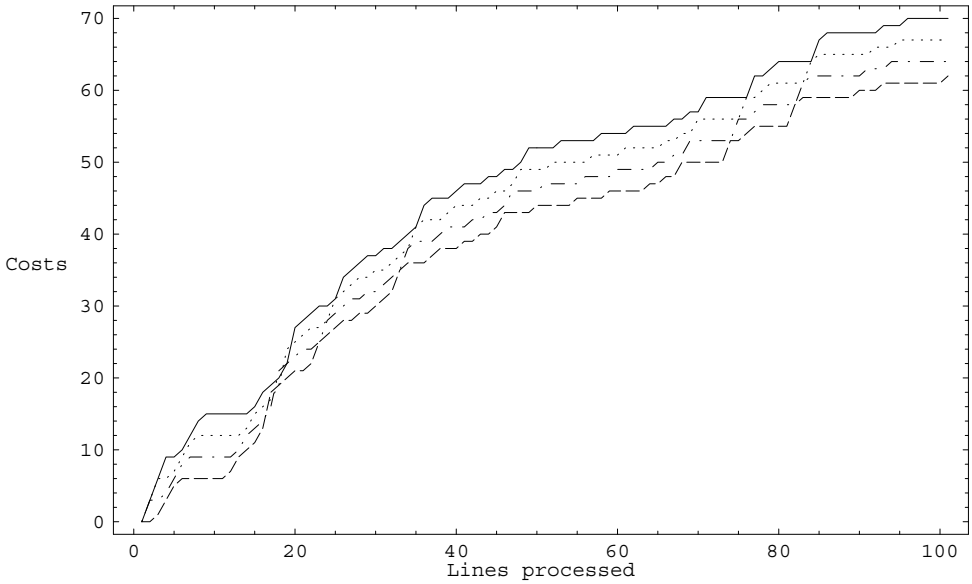


Figure 9.4 : Influence des entités apprises sur un scénario AUD.

9.4 Expériences

Cette section présente nos expériences conduites avec des analyseurs utilisés en reconnaissance de texte, et qui offrent une forme d'apprentissage incrémental. L'objectif est double. D'une part, il s'agit de montrer l'utilité de notre modèle de coûts formel. D'autre part, nous cherchons à mesurer concrètement l'avantage de l'approche assistée, en fixant les conditions d'utilisation.

9.4.1 Méthodologie

Les avantages de l'approche AMD sur BATCH ont déjà été motivés [194]. Il nous reste à évaluer les bénéfices potentiels de l'approche libre AUD. En fait, nous nous limiterons à esquisser les tendances sur la base de deux paquets de test, où nous supposerons que le coût est directement proportionnel au nombre d'erreurs dans l'entité ($\lambda = 0$, $\alpha = \beta = 1$, et donc $C(e[M^t] \S\ e_1[L^t]...e_n[L^t]e[A^t]) = |W^t_{e,<e_1...e_n>}|$).

Soit $\mathcal{P}_{good} \subset \mathcal{P}_{aud} = \{P_i$ tel que $C(P_i) \leq C(P_{amd})\}$. Voici quelques affirmations qu'il s'agit de vérifier :

- $|\mathcal{P}_{good}| \gg 0$: il y a de nombreux motifs meilleurs que les versions AMD;
- $C(P_{opt}) \ll C(P_{amd})$: les meilleurs motifs coûtent beaucoup moins que les versions AMD;
- $\mathcal{P}_{good}$ contient des motifs «intuitifs», au sens où l'utilisateur peut raisonnablement les trouver avec du bon sens pratique; c'est un point crucial, mais nettement plus difficile à quantifier.

9.4.2 Evaluation des coûts avec *ApOFIS*

Le système d'identification de fontes *ApOFIS* [213] offre des fonctionnalités étendues d'apprentissage. Nos expériences ont consisté à calculer le coût de quelques motifs possibles pour produire un document correctement étiqueté par les fontes.

L'échantillon de test (cf. figure 9.5) représente une liste de références, chacune composée de trois informations formattées avec des fontes différentes : un nom en gras, une affiliation en italique, et une adresse en fonte normale. Même si ces informations sont toujours présentes, leur position et leur longueur sont variables, notamment pour l'affiliation qui peut ou non figurer sur une ligne séparée. Puisque les changements de fonte peuvent apparaître à l'intérieur des lignes de texte, un traitement entièrement automatique serait forcé de prendre le mot comme unité d'analyse, alors qu'un pilotage manuel pourrait adapter la granularité au cas par cas.

Le tableau 9.7 reproduit les coûts de reconnaissance obtenus avec différentes stratégies : un scénario par lots (P_{batch}), trois variantes assistées et dirigées par la machine (P^i_{amd}), et trois variantes pilotées par l'utilisateur (P^i_{aud}). Pour P^i_{amd}, nous supposons que les résultats sont soumis par blocs à l'opérateur, qui peut alors

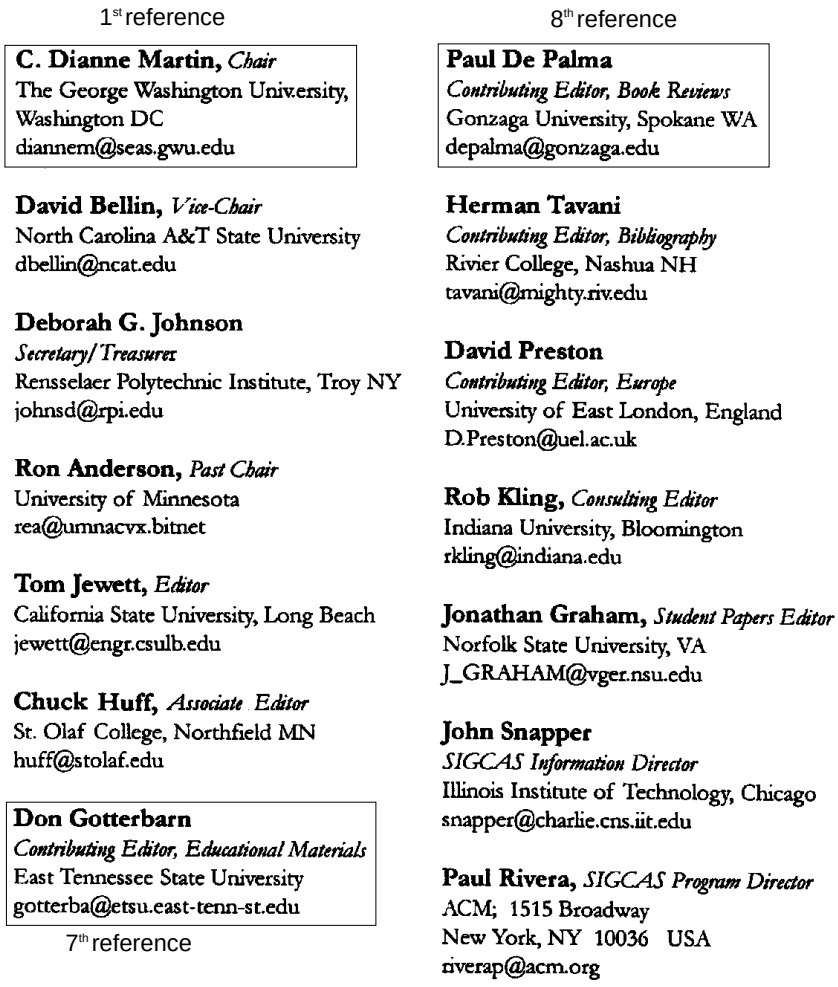


Figure 9.5 : Document de test pour $ApOFIS$ (tiré d'une revue ACM-SIGCAS).

corriger les mots, et éventuellement en demander l'apprentissage. Dans le motif P^1_{amd}, chaque correction est apprise, alors que dans P^2_{amd} et P^3_{amd}, seul un sous-ensemble de mots erronés sont appris.

Cette expérience confirme que des scénarios intuitifs P^i_{aud} se révèlent moins coûteux que des approches dirigistes. Ainsi le scénario P^3_{aud} génère respectivement une économie d'environ 27% et 39% par rapport à P^2_{amd} et par rapport à P_{batch}.

Motifs (P)		Coûts			
		$C^A(P)$	$C^M(P)$	$C^L(P)$	$C(P)$
P_{batch}	$Words[A]$ $Words[M]$	1	68	-	69
P^1_{amd}	$Blocks[A \cup (AML)]$	14	23	23	60
P^2_{amd}	$Blocks[A \cup (AML) \cup (AM)]$	14	29	15	58
P^3_{amd}	$Blocks[A \cup (AML) \cup (AM)]$	14	31	18	63
P^1_{aud}	$Blocks[A \cup M \cup L]$	13	27	9	49
P^2_{aud}	$Blocks[A \cup M \cup L]$	13	23	20	46
P^3_{aud}	$Blocks[A \cup M \cup L]$	13	23	16	42

Tableau 9.7 : Coûts de session avec $ApOFIS$.

9.4.3 Evaluation des coûts avec ScanWorX

ScanWorX [29] est un OCR assez représentatif [176] qui offre une possibilité d'accumuler des connaissances durant le traitement. Dans nos expériences, nous nous concentrons sur le nombre d'erreurs cumulé, sans

		Echantillons				
		V008	V00E	WOUA	WOU9	OFR1
Motifs	$P_{worst'}$	102	130	52	62	61
	P_{batch}	60	105	50	54	43
	P_{amd}	55	78	42	42	41
	$P_{opt'}$	43	54	31	30	25

Tableau 9.8 : Coûts d'édition avec ScanWorX.

pénaliser le lancement des requêtes. Le volume de test s'est limité à 10 pages d'articles scientifiques, extraites de la base de données UW [127], et traitées au niveau du bloc. Pour chaque échantillon, nous avons d'abord produit une version idéale du texte associé à la segmentation, puis appliqué les expériences concrètes suivantes :

- une session complète sans apprentissage, suivant le schéma BATCH;
- une session AMD en traitant les blocs par ordre de lecture;
- un traitement de tous les couples de blocs suivant le motif local $e_i[L^c]e_j[A^c]$, afin de remplir une matrice indiquant l'effet d'apprentissage de chaque bloc sur chaque autre ($|W^c_{e_i,<>} \setminus W^c_{e_i,<e_j>}|, \forall e_i, \forall e_j$); cette expérience permet de juger la variabilité des coûts dans les approches AUD.

Notons que nous n'avons pas tenu compte des erreurs de segmentation en mots, puisqu'il n'y a pas d'apprentissage correspondant supporté par ScanWorX. Par ailleurs, nos résultats ne renseignent pas vraiment sur la qualité du logiciel; nous n'avons par exemple pas cherché à optimiser les paramètres de configuration.

La première impression que nous laissent les expériences est une sensibilité aiguë du processus de reconnaissance au paramétrage. Le taux d'erreur est parfois modifié de manière étrange, et plusieurs essais sont nécessaires pour anticiper les réactions du système.

Le tableau 9.8 compare les coûts d'édition de quelques motifs, pour les échantillons les plus caractéristiques. Nous constatons d'abord que AMD est toujours plus avantageux que BATCH, ce qui signifie que l'effet d'apprentissage est en moyenne positif. Les motifs $P_{opt'}$ et $P_{worst'}$ ont été dérivé de la matrice $W^c_{e_i,<e_j>}$, de façon à exploiter les effets d'apprentissage extrêmes entre blocs; ils dénotent ainsi une borne inférieure et supérieure aux motifs AUD de coût maximal respectivement minimal. On peut observer qu'il y a une amélioration substantielle à gagner sur les approches systématiques : les session du type $P_{opt'}$ génèrent une économie de 21-29% sur les motifs P_{amd}, eux-mêmes meilleurs que les motifs P_{batch}.

L'analyse détaillée de la matrice $W^c_{e_i,<e_j>}$ confirme un autre phénomène intuitif : l'apprentissage est plus bénéfique entre entités de la même fonte. Cela donne une motivation pour utiliser ScanWorX de manière monofonte, c.-à-d. en séparant différents fichiers de connaissances selon la fonte.

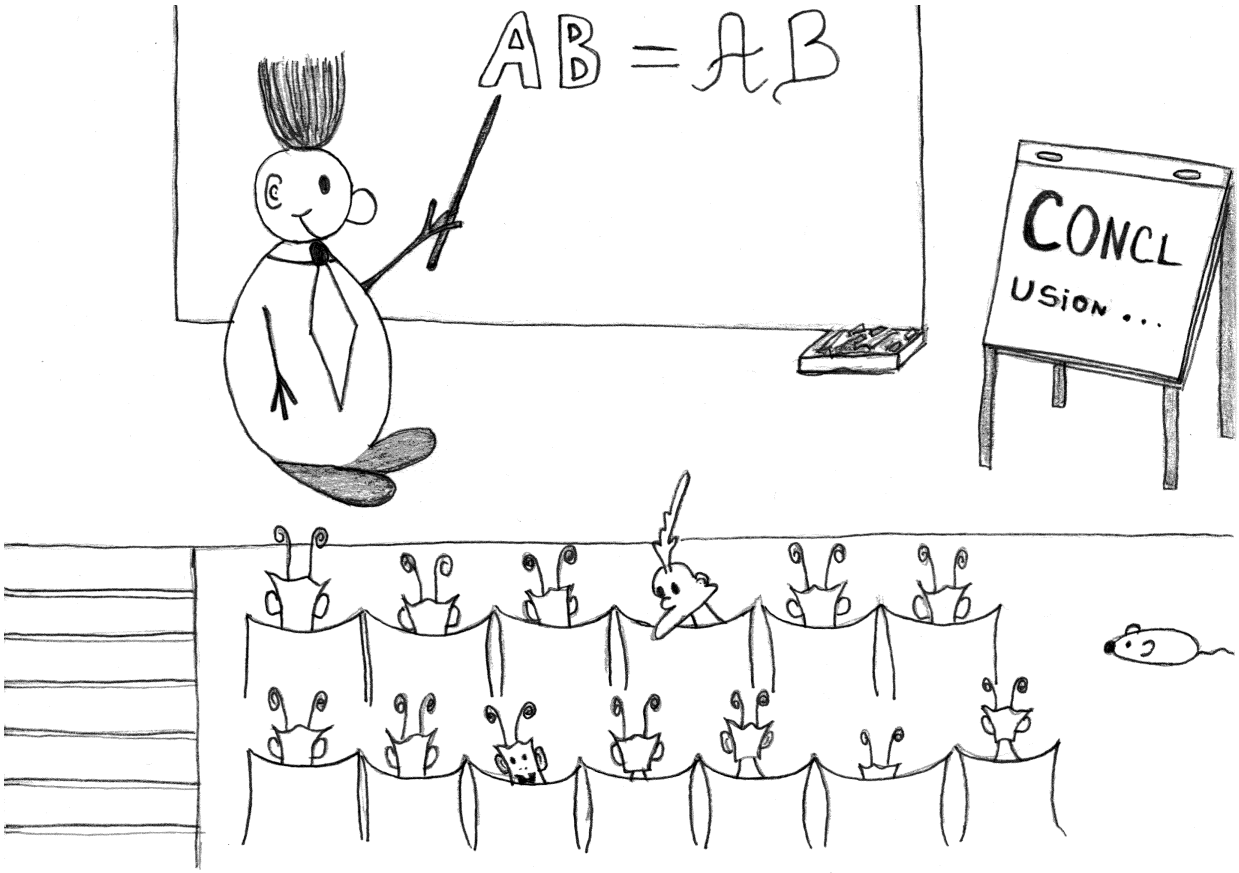
Conclusion

En reconnaissance des documents, l'évaluation des performances d'un système joue un rôle important. D'une part, elle permet de juger la qualité des outils réalisés, à la fois entre eux et par rapport à un outil parfait. D'autre part, le processus d'évaluation met en évidence les aspects de la méthode qui méritent d'être raffinés, et les parties du programme où doivent encore porter les efforts d'optimisation.

Les modèles de coûts traditionnels, basés sur une distance d'édition entre les résultats d'analyse et la forme idéale, ne tiennent compte ni de l'adaptabilité des analyseurs, ni de l'influence de l'utilisateur durant le schéma d'analyse. Pour présenter la problématique d'une manière formelle, nous avons proposé une notation qui permet d'exprimer ces deux phénomènes. Notre modèle a été utilisé pour simuler le comportement d'analyseurs; en observant ces simulations, on se rend mieux compte des avantages de l'apprentissage incrémental. Nous avons également effectué des expériences avec deux outils d'analyse opérationnels. Nos observations sont les suivantes : (i) l'apprentissage incrémental diminue sensiblement les coûts de reconnaissance; (ii) les scénarios dirigés par la machine n'engendrent pas un coût minimum; (iii) avec l'expérience, l'utilisateur tend à diriger une session de travail de façon efficace.

Chapitre 10

Conclusions, perspectives et bilan



Au terme de notre étude, qui a exploré diverses conséquences de l'approche assistée en reconnaissance de documents, ce chapitre expose les conclusions générales du travail.

Le projet *CIDRE*, dans lequel s'insèrent nos travaux, nourrit le dessein ambitieux de réviser toute la problématique de la reconnaissance de documents, en focalisant la discussion sur le rôle de l'utilisateur humain. Dans cette optique, nous nous sommes intéressés aux nombreux problèmes d'architecture logicielle soulevés par l'approche assistée. En explorant les multiples moyens de mieux combiner les compétences de l'homme et de la machine, nous avons proposé des solutions originales sur plusieurs sujets : caractérisation du dialogue homme-machine, modélisation des données et du contrôle dans les systèmes de reconnaissance assistée, développement d'une plateforme logicielle intégrée, conception et réalisation d'outils d'analyse, modélisation des coûts de reconnaissance.

Outre la revalorisation de la main-d'oeuvre, qui constitue le fondement de nos réflexions, on peut relever trois caractéristiques qui imprègnent et empreignent toute notre démarche :

- *Orientation vers les nouveaux besoins :* nous nous sommes efforcés de garder à l'esprit les nouvelles applications que devra servir l'analyse d'image de documents, de manière à préparer le terrain en conséquence.
- *Inspiration des systèmes multi-agents :* bien que nous ne soyons pas des experts qualifiés en intelli-

gence artificielle distribuée, nous nous sommes constamment inspirés des systèmes multi-agents. On y aborde en effet des questions essentielles sur les phénomènes de coopération, qui jouent un grand rôle en reconnaissance assistée. C'est pourquoi toute notre architecture logicielle est organisée selon la philosophie multi-agents.

- *Intégration des ressources informatiques disponibles :* tout au long de notre démarche, nous avons essayé de profiter de ce qui existe déjà, et de le mettre au service du projet *CIDRE*. Ce souci de réutilisabilité touche différents plans : gestion des fontes, programmation distribuée, outils d'analyse existants, formats standards pour les données, bases de données d'images de documents.

La section 10.1 établit l'inventaire des différentes facettes de notre contribution. Les principaux enseignements qui ressortent de nos discussions sont exposés dans la section 10.2. La section 10.3 jette un regard critique sur les solutions que nous avons proposées, et amorce concrètement une série de travaux qui restent à faire. Enfin, la section 10.4 dégage un bilan très général de tout notre travail.

10.1 Résumé des contributions scientifiques

Ce travail s'inscrit dans le cadre du projet *CIDRE*, et a développé l'idée de reconnaissance de documents assistée. Comme annoncé précédemment, l'accent a été mis sur la conception et la réalisation d'une architecture logicielle capable de satisfaire une grande variété d'applications de reconnaissance. Voici un résumé de nos principales contributions scientifiques :

- Des prévisions ont été émises quant à l'évolution des besoins en reconnaissance de documents. Cela nous a permis d'apporter des critiques constructives sur les systèmes de reconnaissance existants, notamment sur le plan du rôle de l'utilisateur, de l'architecture logicielle, ou de l'adaptabilité des analyseurs.
- Quelques idées sur la conduite du dialogue homme-machine ont été développées, dans le contexte de la reconnaissance de documents assistée. L'étude a tenté de concrétiser la notion d'environnement coopératif, p. ex. en présentant plusieurs styles de dialogue, ainsi que quelques conseils ergonomiques.
- Une organisation des données a été présentée pour englober toutes les informations que doit gérer le système de reconnaissance. Une partie de la solution proposée a été implémentée en utilisant le standard DAFS [174].
- Une architecture logicielle distribuée a été modélisée. Cette découpe conceptuelle du système de reconnaissance est inspirée des paradigmes multi-agents, et intègre l'utilisateur comme un participant à part entière.
- Une plateforme d'implémentation a été réalisée; elle respecte de près la philosophie multi-agents, exploite le parallélisme dans une utilisation en réseau, et offre le niveau d'expression adéquat quant à la programmation de l'interface utilisateur.
- La conception d'outils d'analyse a été considérée sous un angle critique, afin d'illustrer par quels moyens on peut augmenter l'utilité des techniques de reconnaissance. Des exemples concrets ont montré comment exploiter la gestion des fontes, telle que la supportent nos environnements informatiques, au profit de la reconnaissance de texte.
- Quelques analyseurs inédits ont été réalisés et testés, notamment dans les domaines de l'OCR, de l'identification des fontes et de la segmentation. Ces outils d'analyse sont conçus pour s'intégrer au mieux dans le reste de l'architecture logicielle. Les résultats se révèlent prometteurs.
- Un modèle d'évaluation original a été proposé dans le but de tenir compte de l'adaptabilité du système de reconnaissance, ainsi que de l'influence de l'utilisateur humain. Ce modèle de coûts a été utilisé aussi bien pour des simulations que dans des expériences avec des outils d'analyse existants.

10.2 Leçons à tirer

Nos travaux autour de l'architecture logicielle nous ont apporté un certain nombre de convictions. La liste suivante synthétise les messages les plus importants qui ressortent de notre étude :

- A l'avenir, il sera difficile de mettre une équipe de développement derrière chaque application de réingénierie de documents. Les logiciels de reconnaissance devront être adaptatifs. Les conséquences de cette évolution sont multiples et profondes.

- L'architecture logicielle devient un élément essentiel du système de reconnaissance. La tendance est à repousser les décisions jusqu'en phase d'exécution, et c'est à l'architecture de supporter cet aspect dynamique. C'est ainsi que dans la philosophie *CIDRE*, l'architecture est par nature ouverte et interactive.
- Les systèmes paramétrés par un modèle de document ne devraient plus faire l'hypothèse que la description formelle des structures de documents est un paramètre invariable. Au contraire, le modèle doit pouvoir s'enrichir au cours du temps. Cette remarque s'applique d'ailleurs à toute la configuration du système et de ses analyseurs automatiques : c'est en phase d'exploitation que le logiciel doit ajuster son paramétrage.
- L'approche multi-agents n'offre pas de solution miracle, mais ses paradigmes s'appliquent bien pour modéliser un système de reconnaissance de documents assisté.
- Les techniques de programmation avancées (programmation concurrente ou distribuée, langages de script, etc.) permettent de mieux exploiter nos environnements informatiques; ils peuvent améliorer l'efficacité, la convivialité ou l'expressivité des programmes.
- Pour concevoir un système de reconnaissance de documents assisté, nous préconisons des techniques d'analyse simples et la concentration des efforts sur l'intégration des composants dans l'environnement interactif. Cette voie est plus prometteuse qu'une interface hermétique qui cache une sur-optimisation des détails algorithmiques.
- L'adaptabilité est une qualité essentielle des outils d'analyse de documents, qui échappe aux modèles d'évaluation classiques.
- Une gestion de fontes standardisée existe maintenant dans tous nos environnements informatiques. Il n'y a aucune raison de s'en priver si cela peut faciliter la reconnaissance de texte imprimé.
- L'interdisciplinarité continuera à servir de moteur d'innovation en analyse d'images de document. Nous l'avons constaté par exemple dans l'apport de la gestion des fontes.

10.3 Critiques et extensions

La présente étude a abordé un thème très vaste, dont chaque facette pourrait faire l'objet d'une analyse plus approfondie. Nous regroupons ici une liste des points qu'il serait particulièrement souhaitable de développer davantage :

- Nous avons évité de biaiser l'architecture par les spécificités d'un type de document. Il s'agit maintenant de profiter de cet avantage, en réalisant une application de reconnaissance de documents en vraie grandeur, grâce à notre plateforme et aux principes sous-jacents. Cela permettra de mettre en évidence la pertinence de nos idées.
- La reconnaissance de documents assistée mériterait une étude plus poussée sur l'ergonomie. Ainsi nous pensons qu'il manque encore une modélisation rigoureuse du fonctionnement du système, ainsi que des évaluations, dans le terrain, d'une variété de prototypes. Face à ce problème ouvert, nous avons estimé que la conception d'une plateforme logicielle adéquate était une étape préalable indispensable, susceptible de former une base de discussion commune.
- La proposition sur l'organisation des données a été incomplètement réalisée. D'une part, la gestion des structures de documents [38] dans *CIDRE* ne s'est stabilisée que très récemment. D'autre part, il y a forcément une part des données qui dépend de l'application ciblée, de même que des analyseurs utilisés. Toutefois, le modèle conceptuel nous semble reposer sur des bases saines, et nos développements attestent de la pertinence des choix de l'implémentation. De plus, notre approche modulaire est suffisamment souple pour que les retouches éventuelles puissent être envisagées avec confiance.
- Dans notre utilisation du terme multi-agents, nous avons mis l'accent sur la puissance d'abstraction. L'intelligence artificielle distribuée ne s'arrête pas là, et il serait très intéressant d'approfondir encore les synergies entre les deux disciplines. Nous pensons que certains problèmes en reconnaissance de documents sont de nature à profiter des progrès en systèmes multi-agents. Par exemple, les problèmes de segmentation pourraient se modéliser en terme de forces [133] exercées par les constituants. De même, la coopération ne se limite pas aux formes de vote, telles qu'on les utilise déjà pour combiner plusieurs OCR; des concepts comme la négociation [121] ou la coopération implicite [54] finiront peut-être par être exploités dans les algorithmes de reconnaissance.

- Les techniques d'analyse proposées (cf. chap. 8) doivent encore être explorées plus attentivement. Les tests effectués jusqu'ici ont surtout une valeur indicative. Dans le contexte de *CIDRE*, il faudra notamment multiplier les expériences sur l'apprentissage incrémental.
- Le modèle de coûts du chapitre 9 est une première ébauche, et son impact réel reste à démontrer. La notation serait peut-être moins compliquée si on établissait le lien avec les modèles d'interface homme-machine.

Afin d'amorcer explicitement de nouveaux sujets de réflexion, nous proposons une série de travaux qui restent encore à faire, dans le prologement des idées débattues dans cette thèse. La matière est organisée ici sous forme de brefs énoncés.

1. *OCR d'écran.* Développer un utilitaire qui se greffe au niveau du gestionnaire de fenêtrage, et qui pilote un OCR sur une portion quelconque de l'écran. Commencer par traiter le cas des affichages noir-blanc.
2. *Masques ternaires pour l'OCR.* Développer une méthode robuste pour construire le squelette et l'enveloppe de chaque caractère d'une fonte. Essayer d'exprimer formellement ce que doit être un «bon» ensemble de masques. La méthode doit fonctionner sur toutes les fontes (petites, grandes, fines, grasses, exotiques).
3. *Parallélisme et accélération.* Concevoir des expériences permettant d'évaluer l'accélération que peut offrir le parallélisme tel que le prévoit notre architecture. Dans une situation donnée, mesurer les temps d'exécution en variant la répartition physique des traitements.
4. *Environnement de reconnaissance basé sur Thot.* Définir un modèle de document Thot [173] (schémas de structure et de présentation) qui permette de représenter les documents à différents stades du processus de reconnaissance. Ce modèle doit accepter initialement la séquence des images des pages, puis la découpe physique. Une ligne de texte doit ensuite pouvoir être remplacée par le texte et la fonte reconnus. Etendre l'éditeur Thot pour pouvoir manipuler plus facilement des échantillons conformes à ce modèle, en spécifiant par là-même un environnement manuel de reconnaissance de document.
5. *Apprentissage incrémental avec ApOFIS.* L'outil de reconnaissance de fontes *ApOFIS* possède déjà des fonctionnalités adaptatives. En partant d'une base de connaissances générée sur des images idéales, mesurer la précision sur des pages scannées, à mesure qu'on adapte les connaissances par apprentissage incrémental.
6. *Prolongement du modèle de coûts.* Trouver une expression formelle du meilleur scénario de reconnaissance, c.-à-d. celui qui optimise le recours à l'apprentissage incrémental dans une situation donnée. Imaginer une démarche qui permette de s'en approcher dans des expériences.
7. *Notations du modèle de coûts.* Transformer les notations de notre modèle de coûts pour les exprimer dans le formalisme UAN (User Action Notation) [83]. Construire un prototype qui permette de tracer dans ce formalisme l'enchaînement des opérations exécutées. Envisager de paramétrer ce prototype par une description de la stratégie d'analyse souhaitée.

10.4 Bilan général

Notre travail a exploré de nombreux sous-problèmes liés à une approche assistée de la reconnaissance de documents. Globalement, notre étude a permis d'esquisser les systèmes de reconnaissance du futur, et de préparer un environnement logiciel adéquat. Dans chaque sujet abordé, nous avons l'impression d'avoir accompli un petit pas vers cet objectif commun. La prochaine grande étape consistera à mettre l'ensemble de nos idées et de nos développements au service d'une application phare.

Certaines de nos contributions tiennent de l'innovation, et d'autres de la synthèse. Certaines idées ont été poursuivies jusqu'à la réalisation d'outils logiciels et à l'expérimentation, alors que d'autres propositions sont restées au stade embryonnaire.

Le choix d'une étude en largeur plutôt qu'en profondeur, un peu étonnant dans une thèse, se justifie par la situation actuelle en analyse d'images de documents : en face de la grande variété des travaux dans cette discipline et de leur relatif cloisonnement, il devient de plus en plus important d'apprendre à intégrer les technologies existantes. Par ailleurs, nous avons cherché à profiter du caractère pluridisciplinaire de

la recherche en reconnaissance de documents, afin de mettre en évidence un certain nombre de synergies possibles.

En regard de l'évolution générale de la société, et notamment du rôle qu'y joue l'informatique, l'approche choisie manifeste une intention de se recentrer sur l'Humain. Il nous semble qu'au-delà des considérations économiques ou scientifiques, c'est bien aux implications sur les activités humaines que doit réfléchir le chercheur.

Finalement, nous espérons que cette étude aura servi à jeter des bases consistantes pour le développement de systèmes de reconnaissance conviviaux, rentables, et aptes à satisfaire les nouveaux besoins en réingénierie de documents.

Bibliographie

[1] Adobe Systems Inc. *PostScript Language*. Addison-Wesley Company, Inc., 1986.

[2] Adobe Systems Inc. *Portable Document Format Reference Manual*. Addison-Wesley, 1993.

[3] A. Agarwal, A. Gupta, K. Hussein, and P. S. P. Wang. Bank Check Analysis and Recognition by Computers. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 24, pages 623–651. World Scientific, 1997.

[4] E. Akpotsui, V. Quint, and C. Roisin. Type Modelling for Document Transformation in Structured Editing Systems. In *Mathematical and Computer Modelling*, pages 1–19. Elsevier Science, 1997.

[5] A. Amin. Arabic character recognition. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 15, pages 397–420. World Scientific, 1997.

[6] J. André and R. D. Hersch. Teaching Digital Typography. *Electronic Publishing: Origination, Dissemination and Design*, 5(2): 79–90, 6 1992.

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

[8] C. Asakawa and T. Itoh. User Interface of a Home Page Reader. In *The Third International ACM SIGCAPH Conference on Assistive Technologies*, Marina del Rey, CA USA, April 1998.
`http: //www.acm.org/sigcaph/assets/assets98`.

[9] R. N. Ascher and G. Nagy. An Integrated Man-Machine System for Reading Printed Text. In *National Electronics Conference*, volume 28, pages 486–489, Chicago, December 1970.

[10] A. Azokly. *Une approche générique pour la reconnaissance de la structure physique de documents composites*. PhD thesis, IIUF-Université de Fribourg, 1995. n. 1105.

[11] S. C. Bagley and G. E. Kopec. Editing Images of Text. *Communications of the ACM*, 37(12): 63–72, Dec 1994.

[12] D. Bainbridge and N. Carter. Paper-based Map Processing. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 22, pages 583–602. World Scientific, 1997.

[13] H. S. Baird. Document Image Defect Models. In H. Baird, H. Bunke, and K. Yamamoto, editors, *Structured Document Image Analysis*, pages 546–556. Springer Verlag, 1992.

[14] H. S. Baird, H. Bunke, and K. Yamamoto. *Structured Document Image Analysis*. Springer-Verlag, 1992.

[15] H. S. Baird and G. Nagy. A self-Correcting 100-Font Classifier. In *SPIE-The international Society for Optical Engineering, Document Recognition*, pages 106–115, San Jose, California, February 1994.

[16] F. Bapst, R. Brugger, and R. Ingold. Towards an Interactive Document Recognition System. Internal working paper 95-09, IIUF-Université de Fribourg, March 1995.

[17] F. Bapst, R. Brugger, A. Zramdini, and R. Ingold. Integrated Multi-Agent Architecture for Assisted Document Recognition. In *DAS'96*, pages 172–188, Malvern, Pennsylvania, October 1996. Reprinted in: Document Analysis II, J. J. Hull and S. L. Taylor (Eds), World Scientific, 1998, pages 301-317.

[18] 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.

[19] F. Bapst and R. Ingold. Using Typography in Document Image Analysis. In R. D. Hersch, J. André, and H. Brown, editors, *Electronic Publishing, Artistic Imaging and Digital Typography (RIDT'98)*, number 1375 in Lecture notes in computer science, pages 240–251, March 1998.

[20] F. Bapst and O. Krone. A Coordination Kernel for Coupling Heterogenous Programming Environments. Internal working paper n. 97-03, IIUF-Université de Fribourg, February 1997.

[21] F. Bapst, A. Zramdini, and R. Ingold. A Scenario Model Advocating User-driven Adaptive Recognition Systems. In *ICDAR'97*, pages 745–748, Ulm-Germany, August 1997.

[22] J. G. P. Barnes. *Programming in Ada*. Addison-Wesley, 1984.

[23] O. Baujard. COALA : Un langage pour la conception de systèmes adaptatifs de résolution de problèmes. Technical report, Laboratoire TIMC/IMAG - Grenoble (France), 1994.
`http: //www-timc.imag.fr/~leloupp/menu.html`.

[24] S. Baumann, M. B. H. Ali, A. Dengel, T. Jäger, M. Malburg, A. Weigel, and C. Wenzel. Message Extraction from Printed Documents – A Complete Solution. In *ICDAR'97*, pages 1055–1059, Ulm-Germany, August 1997.

[25] T. Bayer, U. Bohnacker, and H. Mogg-Schneider. InfoPortLab – An Experimental Document Analysis System. In *Workshop on Document Analysis Systems (DAS'94)*, Kaiserslautern, 1994.

[26] T. Bayer, U. Bohnacker, and I. Renz. Information Extraction From Paper Documents. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 25, pages 653–677. World Scientific, 1997.

[27] J. L. Bentley. *Writing Efficient Programs.* Prentice-Hall, 1982.

[28] D. Benyon and P. Palanque, editors. *Critical Issues in User Interface Systems Engineering.* Springer, 1996.

[29] Beth Paddock and Timothy J. Platt. *ScanWorX API, Programmer's Guide.* Xerox Imaging Systems, Inc., 9 Centennial Drive, Peabody, Massachusetts 01960, 1992.

[30] M. Bever, K. Geihs, L. Heuser, M. Mulhauser, and A. Schill. Distributed Systems, OSF DCE and Beyond. In A. Schill, editor, *International DCE Workshop*, number 731 in LNCS, Karlsruhe, October 7–8 1993. Springer Verlag.

[31] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and Magic Lenses: The See-Through Interface. In *ACM SIGGRAPH'93*, Computer Graphics Annual Conference Series, pages 73–80, 1993.

[32] R. Bippus, V. Märgner, and E. Schütz. An Interactive Document Segmentation and Labeling System Based on a Universal Data Structure. In *DAS'94*, 1994.

[33] D. Blostein and A. Grbavec. Recognition of Mathematical Notation. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 21, pages 557–582. World Scientific, 1997.

[34] S. Bonhomme and C. Roisin. Interactively Restructuring HTML Documents. *Computer Network and ISDN Systems*, 28(7-11): 1075–1084, May 1996.

[35] R. Bradford and T. Nartker. Error Correlation in Contemporary OCR Systems. In *ICDAR'91*, pages 516–524, 1991.

[36] R. Brugger. *A Document Recognition Approach Based on Generalized N-Grams.* PhD thesis, IIUF-Université de Fribourg, 1999. à paraître.

[37] R. Brugger, F. Bapst, and R. Ingold. A DTD Extension for Document Structure Recognition. In R. D. Hersch, J. André, and H. Brown, editors, *Electronic Publishing, Artistic Imaging and Digital Typography (EP'98)*, number 1375 in Lecture Notes in Computer Science, pages 343–354, St-Malo, France, March 1998.

[38] R. Brugger, A. Zramdini, and R. Ingold. Modeling Documents for Structure Recognition Using Generalized n-grams. In *ICDAR'97*, pages 56–60, Ulm-Germany, August 1997.

[39] H. Bunke and P. Wang. *Handbook of Character Recognition and Document Image Analysis.* World Scientific, 1997.

[40] F. Chantemargue, O. Krone, M. Schumacher, T. Dagaeff, and B. Hirsbrunner. Autonomous Agents: from Concepts to Implementation. In *Fourteenth European Meeting on Cybernetics and Systems Research (EMCSR'98)*, pages 731–736, Vienna (Austria), April 1998.

[41] F. R. Chen and D. S. Bloomberg. Extraction of thematically relevant text from images. In *Symposium on Document Analysis and Information Retrieval (SDAIR'96)*, pages 163–178. University of Nevada Las Vegas, 1996.

[42] F. R. Chen and D. S. Bloomberg. Extraction of Indicative Summary Sentences from Imaged Documents. In *ICDAR'97*, pages 227–232, Ulm-Germany, August 1997.

[43] S. Chen. OCR Performance Evaluation Software User's Manual. Technical report, University of Washington, 1993. distributed on the CD-ROM database UW-I.

[44] Y. Chenevoy. *Reconnaissance structurelle de documents imprimés: études et réalisations.* PhD thesis, CRIN-Nancy, Décembre 1993.

[45] L. Cholvy. Fusion de sources d'informations ordonnées en fonction des thèmes. In *9ème congrès en reconnaissance des formes et intelligence artificielle*, volume 2. AFCET France, Janvier 1994.

[46] C. K. Chow. Recognition Error and Reject Trade-Off. In *Symposia on Document Analysis and Information Retrieval (SDAIR'94).* University of Nevada Las Vegas, 1994.

[47] J.-L. Cochard and P. 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.

[48] B. Cooperman. Producing Good Font Attribute Determination Using Error-Prone Information. In L. M. Vincent and J. J. Hull, editors, *Document Recognition IV*, SPIE - The International Society for Optical Engineering, pages 50–57, San Jose, California, February 1997.

[49] Corba. Object Request Broker Architecture. Technical report, Object Management Group, 1993. OMG TC Document 93.7.2.

[50] J. Coutaz. *Interfaces homme-ordinateur.* Dunod informatique, 1990.

[51] J. Coutaz, L. Nigay, and D. Salber. Agent-Based Architecture Modelling for Interactive Systems. In D. Benyon and P. Palanque, editors, *Critical Issues in User Interface Systems Engineering*, chapter 11, pages 191–210. Springer, 1996.

[52] C. Cracknell, A. C. Downton, and L. Du. TABS – A New Software Framework for Document Image Processing, Analysis, and Understanding. In *ICDAR'97*, pages 1001–1005, Ulm-Germany, August 1997. `http: //vasawww.essex.ac.uk/~craccb/`.

[53] A. Cypher, D. S. Kosbie, and D. Maulsby. Characterizing Programming By Demonstration Systems. In A. Cypher, editor, *Watch what I do*, pages 467–484. MIT Press, 1993.

[54] T. Dagaeff, F. Chantemargue, and B. Hirsbrunner. Emergence-based Cooperation in a Multi-Agent System. In *Second European Conference on Cognitive Science (ECCS'97)*, pages 91–96, Manchester (UK), April 1997.

[55] D. Dardailler. The Web Accessibility Initiative. In R. D. Hersch, J. André, and H. Brown, editors, *Electronic Publishing, Artistic Imaging and Digital Typography (RIDT'98)*, number 1375 in Lecture Notes in Computer Science, March 1998. Keynote speech (cf. `http: //www.w3.org/WAI/`).

[56] A. Dengel, R. Bleisinger, R. Hoch, F. Hønes, M. Malburg, and F. Fein. Office-MAID – A System for Automatic Mail Analysis, Interpretation and Delivery. In L. Spitz and A. Dengel, editors, *Document Analysis Systems*, pages 52–75. World Scientific, 1995.

[57] E. Dijkstra. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Communications of the ACM*, 18(8): 453–457, 1975.

[58] G. Dimauro, S. Impedovo, and G. Pirlo. Algorithms for Automatic Signature Verification. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 23, pages 605–621. World Scientific, 1997.

[59] Dimund. Document Image Understanding and Optical Character Recognition (OCR) Information and Resources. Laboratory for Language and Media Processing (LAMP), University of Maryland (cf. `http: //documents.cfar.umd.edu/`).

[60] X. Q. Ding. Machine Printed Chinese Character Recognition. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 11, pages 305–330. World Scientific, 1997.

[61] D. Doermann and S. Yao. Generating Synthetic Data for Text Analysis Systems. In *Fourth Symposium on Document Analysis and Information Retrieval (SDAIR'95)*, pages 449–467, 1995.

[62] D. Dori, D. Doermann, C. Shin, R. Haralick, I. Phillips, M. Buchman, and D. Ross. The Representation of Document Structure: A Generic Object-Process Analysis. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 25, pages 421–456. World Scientific, 1997.

[63] L. Duffy. Analyse de la structure logique d'un document par comparaisons des caractéristiques typographiques. In *Reconnaissance de documents: modèles et méthodes*, ENST-Paris, June 1997.

[64] K. P. Durre and K. W. Glander. Design Considerations for Microcomputer Based Application for the Blind. In *Human jobs and computer interfaces*. North-Holland, Amsterdam, 1991.

[65] D. W. Embley and G. Nagy. Behavioral Aspects of Text Editors. *ACM Computing Surveys*, 13(1): 33–70, 1981.

[66] J. Esakov, D. P. Lopresti, J. S. Sandberg, and J. Zhou. Issue in Automatic OCR Error Classification. In *Symposia on Document Analysis and Information Retrieval (SDAIR'94)*, 1994.

[67] ExperVision, Inc., 3590 North First Street, San Jose, CA 95134-9815. *TypeReader Professionnal*, February 1995. Release 1.0 for MacOS.

[68] A. Feng and T. Wakayama. SIMON – A Grammar-Based Transformation System for Structured Documents. *Electronic Publishing*, 6(4): 361–372, 1993.

[69] D. Flanagan. *Java in a Nutshell*. O'Reilly, 1996.

[70] I. Foster and S. Taylor. *Strand - New Concepts in Parallel Programming*. Englewood Cliffs, Prentice Hall, 1990.

[71] R. Furuta. Concepts and Models for Structured Documents. In J. André, R. Furuta, and V. Quint, editors, *Structured Documents*, pages 7–38. Cambridge University Press, 1989.

[72] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[73] M. D. Garris, J. L. Blue, G. T. Candela, P. J. Grother, S. A. Janet, and C. L. Wilson. NIST Form-Based Handprint Recognition System. Technical Report NISTIR 5959, National Institute of Standards and Technology, January 1997. `http: //www.nist.gov/itl/div894/894.03/databases/defs/nist_ocr.html`.

[74] A. Geist, A. Bueguelin, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, Massachusetts, 1994.

[75] E. Glinert and R. Ladner. A Large Font Virtual Terminal Interface. *Communications of the ACM*, 27(6): 567–572, June 1984.

[76] M. Goedicke and B. Sucrow. Towards a Flexible Software Architecture of Interactive Systems. In D. Benyon and P. Palanque, editors, *Critical Issues in User Interface System Engineering*, chapter 11, pages 211–224. Springer, 1996.

[77] M. Haines and P. Mehrota. Special Issue on Multithreading for Multiprocessors: Guest Editors' Introduction. *Journal of Parallel and Distributed Computing*, 37(1): 1–5, August 1996.

[78] D. Heller. *Motif Programming Manual*. O'Reilly, 1991.

[79] A. Hennig, E. Marongiu, N. Sherkat, and R. J. Whitrow. DART – A Software Architecture for the Creation of a Distributed Asynchronous Recognition Toolbox. In *ICDAR'97*, pages 439–443, Ulm-Germany, August 1997. `http: //www.doc.ntu.ac.uk:81/~amr/`.

[80] E. V. Herwijnen. *Practical SGML*. Kluwer Academic Publishers, 1990.

[81] R. Hipp and M. Cruse. An Introduction To Pthreads-Tcl. `http: //www.hwaci.com/sw/pttcl/pttcl.html`.

[82] O. Hitz. Realization of a Generic Monofont OCR. Master's thesis, IIUF-Université de Fribourg, August 1998.

[83] D. Hix and H. R. Hartson. *Developing User Interfaces – Ensuring Usability Through Product & Process*. Wiley, 1993.

[84] T. Ho, J. J. Hull, and R. Srihari. Decision Combination in Multiple Classifier Systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 16(1): 66–75, 1994.

[85] J. D. Hobby. Matching Document Images with Ground Truth. *IJDAR*, 1(1): 52–61, February 1998.

[86] T. Hu. *New Methods for Robust and efficient recognition of the Logical Structures in documents*. PhD thesis, IIUF-Université de Fribourg, 1994. n. 1076.

[87] T. Hu, K. Marukawa, Y. Shima, and H. Fujisawa. A Prototype for Extracting Logical Elements From Tables of Content of Journals. In *DAS'96*, pages 388–414, Malvern, Pennsylvania, October 1996.

[88] A. Hunt. comp.speech Frequently Asked Questions WWW Site. `http: //www.speech.su.oz.au/comp.speech/`.

[89] S. Inglis. SOCR - A Freely Available OCR System Written in C/C++. `http: //www.socr.org`.

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

[91] R. Ingold and J.-L. Cochard. Le projet escroc: Environnement de saisie et de correction de la reconnaissance optique de caractères. In *CNED'94*, pages 257–264, Rouen (France), 1994.

[92] Internet. Tcl-Tk.
`http: //www.tcltk.com`
`http: //www.scriptics.com/`
`http: //web.cs.ualberta.ca/ wade/HyperTcl/`.

[93] ISO. Document Style Semantics and Specification Language (DSSSL) (ISO 10179), 1996.
`http: //occam.sjf.novell.com: 8080/dsssl/dsssl96` .

[94] A. K. Jain and B. Yu. Document Representation and Its Application to page Decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 20(3): 294–308, March 1998.

[95] M. Y. Jaisimha, A. Bruce, and T. Nguyen. DocBrowse: A System for Textual and Graphical Querying on Degraded Document Image Data. In *DAS'96*, pages 581–604, Malvern, Pennsylvania, October 1996.
`http: //www.statsci.com/docbrowse/`.

[96] F. James. Lessons from Developing Audio HTML Interface. In *The Third International ACM SIGCAPH Conference on Assistive Technologies*, Marina del Rey, CA USA, April 1998.
`http: //www.acm.org/sigcaph/assets/assets98`.

[97] G. W. Johnson. *Labview Graphical Programming: Practical Applications in Instrumentation and Control*. McGraw Hill, 1994.

[98] P. Johnson. *Human Computer Interaction – Psychology, Task Analysis and Software Engineering*. McGraw-Hill, 1992.

[99] V. Joloboff. Document Representation: Concepts and Standards. In J. André, R. Furuta, and V. Quint, editors, *Structured Documents*, pages 75–105. Cambridge University Press, 1989.

[100] A. M. Julienne and B. Holz. *ToolTalk and Open Protocols: Inter-Application Communication*. SunSoft Press Englewood Cliffs, 1994.

[101] E. Kant. Interactive Problem Solving using task configuration and control. *IEEE Expert*, 3(4): 36–49, 1988.

[102] T. Kanungo. DDM User Manual. Technical report, Information Science Research Institute (ISRI), University of Nevada Las Vegas (UNLV), January 1997.

[103] P. Karow. *Typeface Statistics*. URW Verlag, Hambourg, 1993.

[104] P. Karow. *Digital Typefaces*. URW Verlag, Hambourg, 1994.

[105] P. Karow. *Font Technology*. URW Verlag, Hambourg, 1994.

[106] M. Kay. The Proper Place of Men and Machines in Language Translation. *Machine Translation*, 12(1-2), 1997. Reprint from 1980 article.

[107] S. Khoubyari and J. J. Hull. Font and Function Word Identification in Document Recognition. *Computer Vision and Image Understanding*, 63(1): 66–74, January 1996.

[108] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas. On Combining Classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 20(3): 226–239, March 1998.

[109] S. Knerr, O. Baret, D. Price, J. C. Simon, V. Anisimov, and N. Gorski. The A2iA Recognition System for Handwritten Checks. In *DAS'96*, pages 431–494, Malvern, Pennsylvania, October 1996.

[110] D. E. Knuth. *The TEXbook*, volume A of *Computer and Typesetting*. Addison Wesley, 1986.

[111] G. E. Kopec. Least-Square Font Metric Estimation from Images. *IEEE Transactions on Image Processing*, 2(4): 510–519, October 1993.

[112] O. Krone. *STL and* PT-PVM*: Concepts and tools for Coordination of Multi-threaded Applications*. PhD thesis, IIUF-Université de Fribourg, 1997. n. 1191.

[113] O. Krone and M. Aguilar. Bridging the Gap: A Generic Distributed Hierarchical Coordination Model for Massively Parallel Systems. In *Proceedings of the '95 SIPAR-Workshop on Parallel and Distributed Computing*, Biel, Switzerland, October 1995.

[114] O. Krone, M. Aguilar, and B. Hirsbrunner. Pt-PVM: Using PVM in a Multi-Threaded Environment. In *Euro-PVM*, Lyon (France), September 1995.

[115] O. Krone, F. Chantemargue, T. Dagaeff, M. Schumacher, and B. Hirsbrunner. Coordinating Autonomous Agents. Internal working paper 97-20, IIUF-Université de Fribourg, 1997.

[116] E. Kuikka and M. Penttonen. Transformation of Structured Documents With the Use of Grammar. *Electronic Publishing*, 6(4): 373–383, 1993.

[117] A. Kundu. Handwritten Word Recognition Using Hidden Markov Model. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 6, pages 157–182. World Scientific, 1997.

[118] M. Kurze. Giving Blind People Access to Graphics (Example: Business Graphics). In *Workshop Nicht-visuelle graphische Benutzungsoberflächen(Software-Ergonomie '95 Workshop)*, Darmstadt, 1995.
`http: //www.inf.fu-berlin.de/~kurze/publications/se_95/swerg95.htm`.

[119] L. Lam, Y.-S. Huang, and C. Y. Suen. Combination of Multiple Classifier Decisions for Optical Character Recognition. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 3, pages 79–102. World Scientific, 1997.

[120] L. Lamport. *Latex, a Document Preparation System*. Addison-Wesley, 1994.

[121] S. Lander. *Distributed Search and Conflict Management Among Resusable Heterogeneous agents*. PhD thesis, University of Massachusetts Amherst, May 1994.

[122] H. Lâasri and B. Maitre. Flexibility and Efficiency in Blackboard Systems: Studies and Achievements in ATOME. In *Blackboards and Applications*. Academic Press, 1989.

[123] P. Lefèvre, C. Felter, and P. Lobbrecht. Reconnaissance de documents: Passage du document papier à l'information électronique. *EPURE*, 58: 15–25, Avril 1998. EDF, Direction des Etudes et Recherches.

[124] P. Lefèvre and F. Reynaud. ODIL: an SGML Description Language of the Layout of Documents. In *ICDAR'95*, pages 480–488, 1995.

[125] B. Lewis and D. J. Berg. *Threads Primer – A Guide to Multithreaded Programming*. Prentice Hall, 1996.

[126] Y. Li, M. Lalonde, E. Reiher, J.-F. Rizand, and C. Zhu. A Knowledge-Based Image Understanding Environment for Document Processing. In *ICDAR'97*, pages 979–983, Ulm-Germany, August 1997.
`http: //www.crim.ca/sbc/cime/` .

[127] J. Liang, J. Ha, R. Rogers, I. Phillips, R. M. Haralick, and B. Chanda. The Prototype of a Complete Document Image Understanding system. In *DAS'96*, pages 131–154, Malvern, Pennsylvania, October 1996.

[128] J. Liang, R. Rogers, R. M. Haralick, and I. Phillips. UW-ISL Document Image Analysis Toolbox: An Experimental Environment. In *ICDAR'97*, pages 984–988, Ulm-Germany, August 1997.

[129] S. Liang, M. Shridhar, and M. Ahmadi. Segmentation of Touching Characters in Printed Document Recognition. *Pattern Recognition*, 22(4): 347–350, 1989.

[130] H. W. Lie and B. Bos. Cascading Style Sheets, level 1 – W3C Recommendation, December 1996.
`http: //www.w3.org/TR/REC-CSS1#ref6`.

[131] D. Lopresti and J. Zhou. Document Analysis and the World Wide Web. In *DAS'96*, pages 651–671, Malvern, Pennsylvania, October 1996.

[132] D. Lopresti, J. Zhou, G. Nagy, and P. Sarkar. Spatial Sampling Effects in OCR. In *ICDAR'95: Third International Conference on Document Analysis and Recognition*, pages 309–314, Montreal, Canada, August 1995.

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

[134] P. Maes. Modeling Adaptive Autonomous Agents. *Artificial Life Journal*, 1(1-2): 135–162, 1994.

[135] T. W. Malone and K. Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1): 87–119, March 1994.

[136] M. J. McLennan. Object-Oriented Programming with [incr Tcl] and Building Mega-Widgets with [incr Tk]. Technical report, Bell Labs Innovations for Lucent Technologies, 1247 S.Cedar CrestBlvd, Allentown PA 18104, 1996.
`http: //www.tcltk.com/itcl/`.

[137] P. A. McWilliams. *Personal Computers and the Disabled.* Quantum Press / Doubleday, 1984.

[138] D. Megginson. *Structuring XML Documents (Extensible Markup Language).* Open Information Management. Prentice Hall, March 1998.

[139] F. Mondada, E. Franzi, and P. Ienne. Mobile Robot Miniaturization: a Tool for Investigation in Control Algorithms. In *ISER'93*, Kyoto, October 1993.

[140] R. Monroe, A. Kompanek, R. Melton, and D. Garlan. Architectural Styles, Design Patterns, and Objects. *IEEE Software*, 14(1): 43–52, January-February 1997.

[141] R. A. Morris. Classification of Digital Typefaces Using Spectral Signatures. *Pattern Recognition*, 25(8): 869–876, 1992.

[142] V. F. Märgner, P. Karcher, and A.-K. Pawlowski. On Benchmarking of Document Analysis Systems. In *ICDAR'97*, pages 331–336, Ulm-Germany, August 1997.

[143] P. Mulhem and L. Nigay. Interactive Information Retrieval Systems: From User Centered Interface Design to Software Design. In *ACM-SIGIR'96*, SIGIR Forum, pages 326–334, Zurich, August 1996.

[144] C. Musciano and B. Kennedy. *HTML (Hyper-Text Markup Language) – The Definitive Guide.* O'Reilly, 2nd edition edition, 1997.

[145] G. Nagy. The Application of Nonsupervised Learning to Character Recognition. In L. N. Kanal, editor, *IEEE Workshop on Pattern Recognition*, pages 391–398, Puerto Rico, 1968. Thompson Book Company.

[146] G. Nagy. At the Frontiers of OCR. *Proceedings of the IEEE*, 80(7): 1093–1100, July 1992.

[147] G. Nagy. Document Image Analysis: Automated Performance Evaluation. In A. L. Spitz and A. Dengel, editors, *International Association for Pattern Recognition Workshop on Document Analysis Systems*, pages 137–155. World Scientific, 1995.

[148] 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.

[149] G. Nagy. Conference Report of ICDAR'07. In *ICDAR'97*, pages 1112–1112, Ulm-Germany, August 1997. Invited talk.

[150] G. Nagy. DIA, OCR, and the WWW. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 28, pages 729–754. World Scientific, 1997.

[151] G. Nagy and S. Seth. Modern Optical Character Recognition. *Froelich/Kent Encyclopedia of telecommunications*, 11: 473–531, 1996.

[152] G. Nagy, S. Seth, and M. Viswanathan. A Prototypical Document Image Analysis System for Technical Journals. *IEEE Computer*, 25(7): 10–22, 1992.

[153] G. Nagy, A. Smal, S. Seth, T. Fischer, E. Guthmann, K. Kalafala, L. Li, S. Sivasubramaniam, and Y. Xu. A Prototype For Adaptive Association of Street Names With Streets on Maps. In K. Tombre and A. K. Chhabra, editors, *Graphics Recognition (GREC'97)*, number 1389 in Lecture Notes in Computer Science, pages 302–313. Springer, August 1997.

[154] G. Nagy and Y. Xu. Priming the Recognizer. In *DAS'96*, pages 263–281, Malvern, Pennsylvania, October 1996.

[155] G. Nagy and Y. Xu. Automatic Prototype Extraction for Adaptive OCR. In *ICDAR'97*, pages 278–282, Ulm-Germany, August 1997.

[156] G. Nagy and Y. Xu. Bayesian Subsequence Matching and Segmentation. *Pattern Recognition Letters*, 18(11-13): 1117–1124, November 1997.

[157] T. A. Nartker. Benchmarking DIA Systems. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 31, pages 801–820. World Scientific, 1997.

[158] Neurotalker. Neurotalker. International Neural Machines inc.
`http: //www.ineural.com/`.

[159] B. Nichols, D. Buttlar, and J. P. Farrell. *Pthreads Programming*. O'Reilly & Associates, Inc., September 1996.

[160] K. Niyogi, S. N. Srihari, and V. Govindaraju. Analysis of Printed Forms. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 18, pages 485–502. World Scientific, 1997.

[161] A. Nye. *Xlib Programming Manual*. O'Reilly, 1990.

[162] L. O'Gorman and R. Kasturi. *Document Image Analysis*. IEEE Computer Society Press, 1995.

[163] M. A. O'Hair and M. Kabrisky. Recognizing Whole Words as Single Symbols. In *ICDAR'91*, pages 350–358, 1991.

[164] N. Oswald and P. Levi. Cooperative Vision in a Multi-Agent Architecture. In *ICIAP'97: International Conference on Image Analysis and Processing*, number 1310 in Lecture Notes in Computer Science, pages 639–646. Springer, September 1997.

[165] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, Inc., 1994. available on ftp sites.

[166] Pad++. Pad++. Human-Computer Interaction Lab, University of Maryland, College Park.
`http: //www.cs.umd.edu/hcil/pad++/` .

[167] F. Parmentier and A. Belaïd. Logical Structure Recognition of Scientific Bibliographic References. In *ICDAR'97*, pages 1072–1076, Ulm-Germany, August 1997.

[168] I. T. Phillips, S. Chen, J. Ha, and R. M. Haralick. English Document Database Design and Implementation Methodology. In *Symposia on Document Analysis and Information Retrieval (SDAIR'93)*, pages 65–104, 1993.

[169] B. Poirier and M. Dagenais. An Interactive System to Extract Structured Text from a Geometrical Representation. In *ICDAR'97*, pages 342–346, Ulm-Germany, August 1997.

[170] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey. *Human-Computer Interaction*. Addison-Wesley, 1994.

[171] E. Quigley. *Perl by Example*. Prentice Hall, 1995.

[172] V. Quint. Systems for the Manipulation of Structured Documents. In J. André, R. Furuta, and V. Quint, editors, *Structured Documents*, pages 39–74. Cambridge University Press, 1989.

[173] V. Quint, H. Richy, C. Roisin, and I. Vatton. *Thot – Manuel utilisateur*. Imag - INRIA, November 1995. V0.95, previously called Grif.
`http: //www.inrialpes.fr/opera/Thot.en.html`.

[174] RAF Technology, Inc. *DAFS Library, Programmer's Guide and Reference*, August 1995.

[175] J. Raviv. Decision Making in Markov Chains Applied to the Problem of Pattern Recognition. *IEEE Transactions on Information Theory*, 3(4): 536–55, 1967.

[176] S. V. Rice, F. R. Jenkins, and T. A. Nartker. The Fifth Annual Test of OCR Accuracy. Technical Report 96-01, Information Science Research Institute (ISRI), University of Nevada Las Vegas (UNLV), April 1996.

[177] L. Robadey and R. Ingold. Détection automatique de pages vides dans un système d'archivage : premières expériences. In *CIFED'98*, pages 286–295, Québec, 1998.

[178] C. Roisin and I. Vatton. Merging Logical And Physical Structures in Documents. *Electronic Publishing*, 6(4): 327–337, 1993.

[179] D. A. Rosenfeld, K. Inque, M. Nivat, P. S. P. Wang, and L. S. Davis. *Parallel Image Analysis – Theory and Practice*. World Scientific, 1995.

[180] P. V. Roy, S. Haridi, P. Brand, G. Smolka, M. Mehl, and R. Scheidhauer. Mobile Objects in Distributed Oz. *ACM Transaction on Programming languages and systems*, 19(5), September 1997.

[181] D. E. Ruddock and B. Dasrathy. Multithreading Programs: Guidelines for DCE Applications. *IEEE Software*, 13(1), January 1996.

[182] P. Sarkar, G. Nagy, J. Zhou, and D. Lopresti. Spatial Sampling of Printed Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 20(3): 344–351, March 1998.

[183] L. Schmutz. SAgA – Un système d'agents autonomes pour l'intelligence artificielle collective. Internal working paper 95-12, IIUF-Université de Fribourg, March 1995.

[184] B. Schneiderman. *Designing the User Interface – Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 1992.

[185] S. Schubiger. BABYLON: Design and Implementation of a Generic Architecture for Coupling Programming Environments. Technical report, IIUF-Université de Fribourg, November 1997. Travail de séminaire.

[186] R. Sennhauser. Improving the Recognition Accuracy of Text Recognition Systems using Typographical Constraints. In *RIDT'94: Third International Conference on Raster Imaging and Digital Typography*, pages 273–282, Darmstadt, Germany, April 1994.

[187] J. H. Shamilian, H. S. Baird, and T. L. Wood. A Retargetable Table Reader. In *ICDAR'97*, pages 158–163, Ulm-Germany, August 1997.

[188] U. Shanker. Autonomous and Mobile Agents in Distributed Network Management and Monitoring System. Technical report, Innovative Network Application – IBM Scientific Center, 18 Vangerowstrasse, Heidelberg 69115 (Germany), April 1998.

[189] H. Shi and T. Pavlidis. Font recognition and contextual processing for more accurate text recognition. In *ICDAR'97*, pages 39–44, Ulm-Germany, August 1997.

[190] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society, 1995.

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

[192] A. L. Spitz. SPAM: A Scientific Paper Access Method. In *DAS'96*, pages 117–130, Malvern, Pennsylvania, October 1996.

[193] S. Srihari, S. Lam, V. Govindaraju, R. Srihari, and J. Hull. Document Understanding: Research Directions. Technical report, CEDAR, State University of New York at Buffalo, 1992.

[194] H. R. Stabler. Experiences With High-Volume, High-Accuracy Document Capture. In *Document Analysis Systems (DAS'94)*, 1994.

[195] K. Stoffel. *Ein neuro-fuzzy gesteuertes Allokationssystem*. PhD thesis, IIUF-Université de Fribourg, 1994. n. 1079.

[196] V. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4): 315–339, December 1990.

[197] K. Taghva, J. Borsack, and A. Condit. Information Retrieval and OCR. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 29, pages 755–777. World Scientific, 1997.

[198] K. Taghva, A. Condit, J. Borsack, J. Kilburg, C. Wu, and J. Gilbreth. The MANICURE document processing system. Technical Report 95-02, Information Science Research Institute, University of Nevada, Las Vegas, March 1995.

[199] N. Talbert. Toward Human-Centred Systems. *IEEE Computer Graphics and Applications*, 17(4): 21–29, 1997.

[200] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[201] K. Tombre and D. Dori. Interpretation of Engineering Drawings. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 16, pages 457–484. World Scientific, 1997.

[202] J. M. Trenkle and R. C. Vogt. Word Recognition for Information Retrieval in the Image Domain. In *Second Annual Symposium on Document Analysis and Information Retrieval (SDAIR'93)*, pages 105–122, UNLV, Las Vegas, April 1993.

[203] US General Services Administration. Managing End User Computing for Users With Disabilities. Technical report, Information resources management service (IRMS), Washington DC, 1989.

[204] P. Wegner. Coordination as Constrained Interaction. In *Coordination languages and models*, number 1061 in Lecture notes in computer science, pages 28–33, April 1996.

[205] P. Wegner. Why Interaction is More Powerful Than Algorithms. *Communications of the ACM*, 40(5): 81–91, May 1997.

[206] J. Willamowski. Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur. In *9ème congrès en reconnaissance des formes et intelligence artificielle*, volume 2. AFCET France, Janvier 1994.

[207] P. Willisson and G. Kuenning. The ISPELL Checker. Technical report, Free Software Foundation, Inc, 1993.

[208] Xerox. Xerox Reading AvantEdge.
`http: //www.xerox.com/xis/readingadvantedge/`.

[209] H. Yamada. Paper-Based Map Processing. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 19, pages 503–527. World Scientific, 1997.

[210] P. Zamperoni. Plus ça va, moins ça va. *Pattern Recognition Letters*, 17: 671–677, 1996.

[211] J. Zhou and D. Lopresti. Extracting Text form WWW Images. In *ICDAR'97*, pages 248–252, Ulm-Germany, August 1997.

[212] T. Ziemke. Adaptive Behavior in Autonomous Agents: A Survey. *PRESENCE*, 7(5), October 1998. Special issue on Autonomous Agents, Adaptive Behavior and Distributed Simulations.

[213] A. Zramdini. *Study of Optical Font Recognition Based on Global Typographical Features*. PhD thesis, IIUF-Université de Fribourg, 1995. n. 1106,
`http: //www-iiuf.unifr.ch/groups/sde/projects/das/apofis.html` .

[214] A. Zramdini and R. Ingold. A Study of Document Image Degradation Effects on Font Recognition. In *ICDAR'95: Third International Conference on Document Analysis and Recognition*, pages 740–743, Montreal, Canada, August 1995.

[215] A. Zramdini and R. Ingold. Optical Font Recognition Using Typographical Features. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 20(8): 877–882, August 1998.

Annexe A

Couplage d'environnements de programmation

Cette annexe présente, de manière extrêmement condensée, notre solution générale au problème du couplage d'environnements de programmation. Cette contribution sort clairement du domaine de la reconnaissance de documents, mais elle est directement issue de nos réflexions sur l'architecture logicielle de *CIDRE*. Nous cherchions à améliorer l'expressivité du couplage entre un moteur d'analyse écrit en C, et une interface graphique écrite en Tcl. Il a ensuite suffi d'un petit effort de généralisation pour élaborer une solution largement applicable. Les différents aspects de notre mécanisme de couplage sont argumentés et développés en détail dans un rapport interne [20].

A.1 Démarche proposée

En général, nous voyons deux moyens pour mélanger différents langages de programmation dans une seule application :

- Définir une construction dans un langage (p. ex. déclarer une procédure Modula-2 avec le mot-clé EXTERNAL), et l'implémenter dans un autre (p. ex. dans le corps d'une fonction C). Le pont sera établi lors de l'édition des liens, ou à travers la génération d'un nouveau noyau (p. ex. un interpréteur Tcl enrichi). Mais cette approche n'est pas générale, parce qu'au moins un des deux environnements doit avoir été explicitement conçu pour accepter l'autre, ce qui est une limitation importante. De ce point de vue, des passerelles devraient être développées pour chaque couple de langages.
- Créer plusieurs programmes qui seront reliés par entrées/sorties durant l'exécution, ce qui est beaucoup plus général, pour autant que le système d'exploitation prévoie une gestion multi-processus.

Notre solution adopte la deuxième approche, et se base essentiellement sur deux paradigmes :

- *Espaces de coordination :* nous abordons le problème de la programmation multi-langages sous l'éclairage de la théorie de la coordination [135, 113]. En mettant l'accent sur l'exécution des processus, nous considérons chaque environnement de programmation comme un espace de coordination : toute instruction exécutée par le programme multi-langage a une portée naturelle, limitée à un seul contexte d'exécution.
- *Construction pivot :* Nous cherchons à définir une construction qui pourrait servir de pivot entre les multiples modèles de calcul. Dans ce but, nous définissons le concept d'*agent* comme une unité qui peut être exprimée par une grande variété de langages (cf. sections suivantes).

Notre approche réunit les notions d'espace de coordination et d'agent au sein d'une métaphore originale. Chaque environnement de programmation est considérée comme un *pays*. Les agents sont les habitants du pays, et le langage définit les lois qui en régissent l'évolution. Cette métaphore a engendré une terminologie[1] qui comprend le *visa*, la *douane*, ou le *bureau des étrangers*.

A.2 Agents – Un protocole général

Nous définissons un agent comme une représentation abstraite d'une entité active, possédant un certain comportement, et capable d'interagir avec un autre agent.

[1] Il est amusant de constater qu'une terminologie assez proche a depuis été proposée dans un projet chez IBM [188].

Naming protocol	`TYPE Agent=...; TYPE AgentKind=...;`
Spawning protocol	`PROC Spawn(IN AgentKind k, OUT Agent new)`

Interaction protocol	`TYPE MsgKind=...; TYPE MsgArgs=...;`
	`PROC Post(IN Agent to, IN MsgKind m, IN MsgArgs a)`
	`PROC Handle(OUT MsgKind m, OUT MsgArgs a)`

Figure A.1 : Notre protocole de spécification d'agents.

La figure A.1 présente notre triple protocole que doit respecter toute réalisation d'agents :

- *Protocole de nommage :* un agent est considéré comme une instanciation d'une classe; il doit y avoir un moyen de désigner à la fois une sorte d'agents et un individu spécifique.
- *Protocole de lancement :* les agents sont générés dynamiquement; cette opération reçoit en entrée une sorte d'agents et produit l'identification de l'individu créé.
- *Protocole d'interaction :* la manifestation de l'influence d'un agent sur le pays et réciproquement est véhiculée par des *requêtes d'interaction*. Les agents ont à disposition un mécanisme pour *poster* une requête, et un autre mécanisme pour les *intercepter*. De plus, une requête d'interaction est définie par une étiquette sémantique, et peut transporter de l'information supplémentaire sous une certaine forme.

Tous les langages n'ont pas été conçus pour respecter cette spécification dans leurs constructions de base, mais nous prétendons qu'ils permettent presque tous d'exprimer le concept. Ainsi, la notion d'agent aura un sens analogue dans les différents pays, quelles que soient les lois locales fixant leurs droits et obligations.

A.3 Expression dépendante du langage de programmation

La spécification d'agent discutée plus haut peut sembler un peu ambitieuse pour être universelle. En fait, nous estimons que l'environnement de programmation doit juste offrir un support pour isoler l'exécution d'un agent (pour préserver son identité), et un moyen pour les activer (à travers le moteur d'exécution). La nature du support est très libre, et peut prendre des formes variées :

- *Mécanisme d'encapsulation :* L'archétype est le concept d'objet, car il définit clairement l'état et le comportement à travers les attributs et les méthodes. Mais on peut aussi utiliser un type abstrait (Modula-2) ou un paquetage (Ada), où chaque appel de routine reflète une partie du comportement. Une simple procédure, qui différencie son exécution suivant le paramètre reçu, peut aussi servir à encapsuler un agent. Pour un langage déclaratif, un groupe de règles logiques ferait l'affaire. Un agent peut aussi être un programme exécutable doté d'une interface en ligne.
- *Mécanisme d'activation :* Une implémentation directe de l'activation passerait par des processus concurrents ou distribués, et une messagerie, comme dans PVM [196] ou ToolTalk [100]. Une gestion d'événements intrinsèque comme dans Tcl-Tk [165] ou X11 [161] peut suffire. Finalement, on peut simuler une boucle principale dans le programme (p. ex. qui attend des commandes en entrée), et activer les agents par un appel de routine, en passant en paramètre une représentation de l'état courant de l'agent.

La figure A.2 montre comment notre protocole d'agent est exprimé dans trois langages de programmation de différente nature : environnement impératif avec messagerie (C avec PT-PVM [114]), langage interprété avec extension objet (itcl [136]), langage logique concurrent (Strand [70]). En fait, il n'est pas étonnant que la notion d'agent semble naturelle, parce que les programmes tendent à être structurés. Notre proposition implique simplement un affinement structurel. Avec notre méthodologie, nous espérons pouvoir coupler des composants vraiment hétérogènes, comme des objets Oz [191, 180] ou Java [69], des tâches Ada [22], des serveurs RPC Unix [30], des widgets X11 [161], des agents Coala [23], des experts Atome [122], des modules Labview [97], voire des robots Khepera [139].

A.4 Spécification du noyau de coordination

Notre démarche consiste à étendre le protocole d'agents décrit précédemment pour permettre la coordination d'agents habitant différents pays (écrits dans différents environnements de programmation). Il s'agit de définir un support pour la création d'agents à distance, et la communication entre agents de différents pays. Nous

Pt-pvm

```
void pageThread(...)
***** INIT BEHAVIOR *****
  ...

do
******** HANDLE *********
 csReceive(...);
 switch(msg_tag) {
 case addBlockMsg:
  ...
********* SPAWN *********
 csSpawn(bkThread,&kid);
********* POST **********
 csSend(kid,flashMsg,&buf);
 ...
 } while(1); }
```

iTcl

```
itcl_class Page { ...
***** INIT BEHAVIOR *****
method constructor{...}{
   ...
}
******** HANDLE *********
method add_block {x y}{
 ...

********* SPAWN ********
set new [Block #auto]
********* POST *********
$new flash 4
  }
 }
```

Strand

```
startPage(Is,...) :-
***** INIT BEHAVIOR *****
 ... ,
 page(Ms,...).

******** HANDLE *********
page([addBk(X,Y)|Is]...):-
 ... ,

********* SPAWN *********
startBlock(NewStream,...),
********* POST **********
NewStream:=[flash(4)|NS1],
  ...
```

Figure A.2 : Notre protocole exprimé dans trois langages differents.

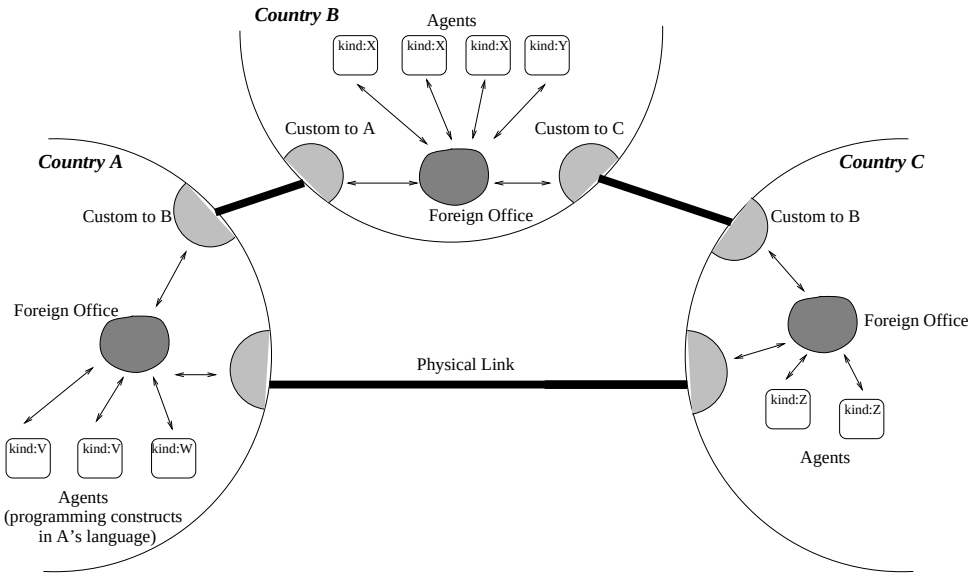


Figure A.3 : Schématisation de notre principe de couplage.

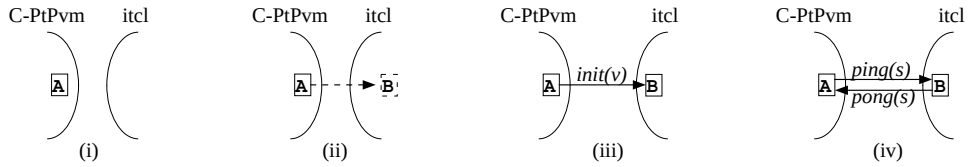
proposons un double mécanisme : (i) offrir deux commandes dédiées au lancement d'agents et au postage de requêtes vers l'étranger; (ii) faire parvenir les requêtes d'interaction sous la forme acceptée par l'opération locale d'interception. Voici quelques motivations en faveur de cette politique :

- Le style de programmation à l'intérieur d'un langage ne devrait pas être perturbé par les nouvelles possibilités de coordination avec l'extérieur. Il est utopique de vouloir redéfinir des constructions de base.
- Notre protocole d'agent est un plus petit dénominateur commun. Rien n'empêche un modèle de programmation d'offrir des fonctionnalités plus riches (p. ex. le filtrage des message reçus, ou l'émission vers des groupes entiers).
- Les opérations de postage et d'interception sont intrinsèquement asymétriques. La première survient durant un contexte d'exécution déterminé; il s'agit de la manifestation d'un comportement volontaire. La deuxième n'est pas censée respecter un schéma convenu à l'avance; il s'agit souvent d'un événement qui va *réveiller* l'agent, et conditionner sa réaction.

La figure A.3 présente l'architecture générale de notre proposition de couplage. Un *office des étrangers* est une instance de contrôle qui sert d'interface entre agents situés dans différents pays. Un agent qui désire interagir avec des agents habitant un autre pays doit utiliser un *visa*. Le visa correspond à un type global qui définit une représentation unique d'un correspondant, utilisée pour l'interaction entre pays.

Declaration	Description
`TYPE AgentKind,Agent`	Agent template+samples (country-dependent)
`TYPE Visa`	String-encoded global IDs for the two types above
`TYPE MsgKind,MsgArgs`	Message passing format (country-dependent)
`CONST String here`	Name of the local country
`PROC ExportAgentKind`	Export an agent class to the outside world
`(IN AgentKind k,`	local template to export
`IN Visa name)`	string respecting «here#here#fantasyName»
`PROC SpawnAbroad`	Create and start agent in other country
`(IN Visa kind,`	template in the remote country
`OUT Visa new)`	the delivered visa for the new born
`PROC SendAbroad`	Send message to a remote agent
`(IN Visa to,`	destinee
`IN MsgKind msg`	msg tag (will be translated)
`IN MsgArgs args)`	related data (will be translated)
`PROC VisaFromAgent`	Deliver a visa if not yet registered
`(IN Agent a, OUT Visa v)`	
`PROC AgentFromVisa`	Get local correspondent identifier
`(IN Visa v, OUT Agent a)`	
`PROC CountryFromVisa`	Get country name where the visa owner lives
`(IN Visa v, OUT String c)`	

Figure A.4 : Spécification de l'office des étrangers.



(i) Startup: countries are set up and A is spawned.
(ii) A (ping) spawns B (pong).
(iii) A sends his visa to B.
(iv) A sends messages and B replies.

Figure A.5 : Scénario du programme ping-pong.

Dans un pays, l'office des étrangers est l'instance centrale qui établit la communication, donc la coordination avec les autres pays, en jouant un double rôle : (i) prendre part à la gestion distribuée des visas; (ii) piloter les ports de connexions entre pays, où seront effectuées les conversions de données.

Dans chaque pays, un agent qui désire profiter des fonctionnalités internationales demande un visa à l'office des étrangers local. Ce visa représente un identificateur unique, qui le désigne univoquement parmi tous les correspondants existants.

La figure A.4 pésente la spécification d'un office des étrangers. Ce module est structuré en trois parties : (i) les types qui permettent de nommer un agent ou une classe; (ii) les primitives propres à la création de nouveaux agents; (iii) les primitives permettant d'interagir à distance.

A.5 Exemple d'utilisation

Les figures A.5 et A.6 présentent un petit programme ping-pong qui utilise notre architecture de couplage. Un agent est écrit en C avec PT-PVM [114]. L'autre est réalisé comme un objet itcl [136].

Cet exemple met aussi en évidence un autre aspect de l'approche multi-langage, à savoir la gestion des dépendances entre différentes familles de codes. Par exemple, il y a une contrainte qui garantit que l'étiquette `handle_ping` définie avec PT-PVM sera traduite dans la chaîne «`handle_ping`» du côté Tcl. L'usage d'expressions littérales empêche de faire contrôler ces contraintes automatiquement. Le programmeur doit être discipliné. Une étape suivante serait l'utilisation d'un langage de définition d'interfaces, afin de générer automatiquement les carcasses de l'application multi-langages. Une ébauche de langage est proposée à la fin de cette annexe.

Excerpt of C-PtPvm part (`ping.c`)

```
#include "pingpong_world.h"
/* This header defines, among others:
 const char AKPong[]="itcl#itcl#AKPong"
 const MSG_TAG init = ...
 const MSG_TAG handle_ping = ...
 const MSG_TAG handle_pong = ...
 void  AKPing(THREAD_ENV *);
  and the foreign office operations.
*/
void AKPing (THREAD_ENV *p) {
 char hisVisa[], myVisa[];  int i;

 SpawnAbroad(AKPong, &hisVisa);
 VisaFromAgent(p->self, &myVisa);
 SendAbroad(hisVisa, init, myVisa);
 for(i=0;i<10;i++) {
  SendAbroad(hisVisa,handle_ping,"Hi");
  csReceive(..., handle_pong, ...);
 }
}
```

Excerpt of itcl part (`pong.itcl`)

```
itcl_class AKPong {
  #   method init        {arg1} {}
  #   method handle_pong {arg1} {}

 inherit pingpong_world
 inherit itcl_foreign_office
  # These classes define, among others:
  #  common handle_pong = "handle_pong"
  # and the foreign office operations.

 protected pingVisa

 method init {v} {
  set pingVisa $v
 }

 method handle_ping {msg} {
  SendAbroad $pingVisa $handle_pong $msg
 }
}
```

Figure A.6 : Sources du programme ping-pong.

A.6 Implémentation

L'implémentation des concepts introduits jusqu'ici n'a été que partiellement accomplie. Nous présentons la méthodologie utilisée, qui nous semble consistante.

Connection physique Nous faisons l'hypothèse que chaque environnement de programmation supporte les communications inter-processus (pipe, sockets, mémoire partagée, signaux, messages, ou simplement entrées/sorties standards). Les canaux sont bidirectionnels, et véhiculent des chaînes de caractères.

Douane Les deux extrémités d'une connexion sont gérées par une douane, responsable d'interfacer l'office des étrangers avec une des frontières. Il s'agit de lire et d'écrire sur le flux, où circulent exclusivement des messages internes de deux types :

- `remote_spawn(IN Visa kind, OUT agent)`;
- `remote_send(IN Visa to, IN MsgKind k, IN MsgArgs a)`

Visa Un visa est une chaîne qui encode un triplet $\alpha\#\beta\#\mu$, où α est le nom du pays où habite le propriétaire, β le nom du pays dont l'office des étrangers a émis le visa, et μ un numéro d'identification. Un office des étrangers qui doit délivrer un visa connaît le pays du propriétaire, et détermine un nouveau μ à l'aide d'un compteur local. De cette manière, μ est unique pour cet office, $\beta\#\mu$ est unique globalement, et α sert à localiser le propriétaire lors de l'acheminement d'un message.

Procédure d'enregistrement distribuée L'association des visa $\alpha\#\beta\#\mu$ avec leur propriétaire est maintenue dans une table gérée par l'office des étrangers du pays α. Nous avons la garantie que chaque agent exporté est enregistré quelque part, bien qu'il n'y ait pas de serveur global de noms.

Dynamisme La création dynamique de liens de communication est une propriété importante de notre spécification de l'office des étrangers. En effet, connaître un visa et un office des étrangers suffit pour poster des messages, et un visa est une simple chaîne de caractères, qui peut être transmise entre agents.

A.7 Concept des jumeaux

Le mécanisme de couplage proposé auparavant ne donne pas de conseil pour organiser l'ensemble d'agents par-dessus les frontières entre pays. Nous allons maintenant proposer une méthodologie qui peut se révéler utile, où les agents sont associés par *jumeaux*.

L'idée des jumeaux consiste à subdiviser une unité conceptuelle en un couple d'individus, de façon à exprimer

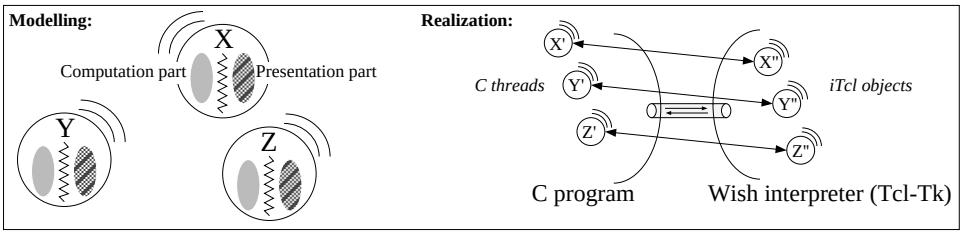


Figure A.7 : Paradigme des jumeaux.

Declaration	Description
`PROC SpawnTwin`	Create the twin in a remote country
`(IN Visa kind,IN Agent me)`	remote template + my local identifier
`PROC SendToTwin`	Communicate with a twin
`(IN Agent me,`	my local identifier
`IN String country,`	remote country name
`IN MsgKind msg,IN MsgArgs args)`	msg tag and data (will be translated)
`PROC GetTwinVisa`	Iterate in the twin ring
`(IN Visa twin1, IN String country,`	known twin + remote country name
`OUT Visa twin2)`	visa of the corresponding twin there

Figure A.8 : Extension de l'office des étrangers pour supporter les jumeaux.

chacune des compétences dans le langage de programmation approprié.

La figure A.7 schématise la notion de jumeau, dans le cas où un programme possède des fonctionnalités interactives et analytiques. Il s'agit d'une situation privilégiée pour le concept de jumeaux, puisqu'il résoud du même coup le lancinant problème du couplage entre le noyau d'une application et son interface graphique.

La figure A.8 présente une extension de la spécification de l'office des étrangers, qui facilite la communication entre jumeaux. Cette solution favorise la lisibilité des sources. Pour l'implémentation, nous proposons que les visas de deux jumeaux ne diffèrent que par le préfixe indiquant le pays qu'ils habitent (ils partagent le même $\beta\#\mu$).

A.8 Perspectives

Loin de prétendre offrir la solution idéale à la programmation multi-langages, nous avons néanmoins accompli un premier pas vers une forme généralisée de couplage entre langages de programmation. Faute de temps, notre mécanisme n'a été implémenté qu'en partie pour quelques langages. Toutefois, nous avons préparé toute l'algorithmique, de sorte qu'un développement complet d'une passerelle ne devrait soulever que des problèmes mineurs.

Certaines extensions peuvent être envisagées. L'exemple du programme ping-pong a permis d'illustrer l'importance des dépendances entre les programmes sources. On pourrait définir un langage servant à spécifier l'architecture générale d'une application en termes d'agents et de pays. Il serait alors possible de développer un générateur automatique de code, afin de garantir l'intégrité des identificateurs par-dessus les frontières de langage. La figure A.9 propose notre ébauche d'un double langage, illustré ici pour l'exemple ping-pong.

Dans son travail de séminaire [185], Simon Schubiger a conçu un langage de spécification multi-langages, réalisé le noyau d'un générateur automatique de code, et validé ce noyau en développant une passerelle complète entre Tcl et C++. Son approche va plus loin que nos idées, grâce à un concept supplémentaire. Chaque agent accessible depuis l'extérieur est représenté dans chaque pays étranger par un *proxy*, sorte de fantôme qui se charge d'interfacer les requêtes d'interaction locales. Cette approche permet de cacher le concept de visa, car toutes les interactions sont exprimées avec les opérations de base offertes par chaque environnement de programmation.

Globalement, notre contribution a permis de faire le lien entre trois domaines de recherche bien distincts : (i) l'interopérabilité, (ii) la théorie de la coordination, et (iii) les systèmes multi-agents.

Agents Definition Language (pingpong.adl)

```
AGENT KIND Ping
  HANDLES init        <visa>
  HANDLES handle_pong <string>

AGENT KIND Pong
  HANDLES handle_ping <string>
```

Countries Definition Language (pingpong.cdl)

```
COUNTRY my_app  IS NEW <C-PtPvm>
  REALIZES AGENT KIND Pong

COUNTRY my_gui  IS NEW <itcl>
  REALIZES AGENT KIND Ping

CONNECTION my_app my_gui IS <unix_pipe>

START my_app
START my_gui
```

Figure A.9 : Ebauches de deux langages pour la définition d'agents et de pays.

Annexe B

Compléments sur notre OCR monofonte générique

Cette annexe présente quelques compléments sur notre paquetage d'OCR monofonte générique, concernant le paramétrage, l'accès aux outils, et le comportement observé lors des tests.

B.1 Paramètres de configuration

Les paramètres de configuration de notre algorithme d'OCR sont résumés dans le tableau B.1 :

- L'alphabet Φ définit la séquence des caractères candidats. Ce paramètre permettra notamment de tenir compte de la langue (caractères accentués), ou du type des champs à analyser (p. ex. seulement des chiffres).
- Les masques d'érosion M_{ero} et de dilatation M_{dil} peuvent dépendre des caractéristiques générales de la fonte. Si le tracé est très fin, on peut renoncer à l'érosion, de peur de perdre toute l'information sur le squelette. S'il est très épais, c'est au contraire la dilatation qui devrait être atténuée.
- Les seuils $d_x, d_y, \alpha, \beta, \gamma, \delta$ sont expliqués dans la section 8.2. On pourrait p. ex. prévoir deux appels à l'OCR, une première fois avec des seuils stricts, et, si le mot est rejeté, une deuxième fois en relachant les contraintes. La deuxième analyse serait plus approfondie, mais nettement plus lente.
- La fonte F est désignée au moyen d'un identificateur valide pour X11. La section 7.3.2 présente nos conventions dans le choix d'alias sous X11.

B.2 Utilitaires

Notre outil de reconnaissance est accessible sous forme d'une librairie en C. Notre paquetage ne repose que sur trois bibliothèques : X11, Tiff, et DAFS [174]. La figure B.1 présente les éléments les plus importants de la spécification.

Le type abstrait «session» englobe toutes les informations qui permettront de lancer une série d'analyses dans les mêmes conditions. Après l'appel à l'OCR, une routine permet de récupérer la chaîne de caractères

Type		**Commentaire**
String	Φ	Alphabet
Image	M_{dil}	Masque de dilatation
Image	M_{ero}	Masque d'érosion
Integer	d_x	Seuil de tolérance horizontal
Integer	d_y	Seuil de tolérance vertical
Float	α	Seuil pour la présélection
Float	β	Seuil pour la mort subite
Float	γ	Ecart relatif maximal sur la dissemblance des bons candidats
Integer	δ	Nombre d'alternatives à garder
String	F	Fonte

Tableau B.1 : Paramètres de configuration de notre outil d'OCR.

```
int mfo_create_font(MFOFont             *font,
                    const char          *fontname,
                    int                  resolution,
                    const unsigned char *alphabet,
                    BFimage              erosion,
                    BFimage              dilation);

int mfo_create_session(MFOSession *session, MFOFont font, BFimage image);

void mfo_set_baseline            (MFOSession *session, int baseline);
void mfo_set_x_tolerance         (MFOSession *session, int x);
void mfo_set_y_tolerance         (MFOSession *session, int y);
void mfo_set_max_miss            (MFOSession *session, double maxmiss);
void mfo_set_preselect_max_miss  (MFOSession *session, double maxmiss);
void mfo_set_branching_score_diff(MFOSession *session, double diff);
void mfo_set_max_branches        (MFOSession *session, int branches);

int          mfo_recognize_word(MFOSession *session,  MFOBox box);
const unsigned char *mfo_result(MFOSession *session, double *word_score);
int             mfo_dafs_result(MFOSession *session, iEntityPtr word);

int mfo_find_baseline(MFOSession *session, MFOBox box);
int    mfo_match_word(MFOSession *session, MFOBox box,
                      const unsigned char *string);
```

Figure B.1 : Extrait de la spécification de notre librairie d'OCR.

reconnue, et la confiance dans le mot trouvé. Le détail des résultats, en particulier les différents chemins évalués, est conservé dans une structure de données. Nous offrons une routine de conversion vers DAFS, et nous pourrions aussi retourner plusieurs alternatives.

Outre la reconnaissance de caractères, notre paquetage offre deux fonctionnalités supplémentaires :

- Une routine permet de *détecter la ligne de base* dans un mot. La technique consiste à essayer de reconnaître le premier caractère, mais avec une tolérance verticale maximale, de manière à tester toutes les hauteurs possibles. Le meilleur candidat nous indique ainsi la ligne de base.
- Nous offrons une routine qui met en correspondance l'image du mot avec une chaîne de caractères déjà connue. Contrairement à l'algorithme de base, cette routine ne considère que le bon candidat à chaque étape. La mesure de dissemblance permet alors d'estimer l'adéquation du couple d'hypothèses (texte + fonte). Lorsqu'on lance cette analyse en variant la fonte, et qu'on classe les résultats suivant la dissemblance, on obtient la fonctionnalité de *reconnaissance de fontes* a posteriori (cf. section 7.3.3).

Plusieurs utilitaires ont été développés pour faciliter l'évaluation de notre méthode d'OCR. Un programme test donnant accès à tous les paramètres a été spécialement développé pour analyser des documents segmentés existants sous forme de fichier DAFS, comme c'est le cas avec notre base de données typographique (cf. section 8.1.2).

La qualité des résultats a été mesurée avec OPE [43], et nous avons écrit quelques scripts en Perl [171] pour générer automatiquement des rapports de performances sous forme de tableaux LaTeX ou de graphes GnuPlot.

B.3 Observations

Notre OCR a notamment été appliqué sur les 174 pages LaTeX scannées de notre base de données (cf. section 8.1.2). Il convient de rappeler que c'est une situation défavorable, dans la mesure où les fontes LaTeX ne correspondent pas aux fontes X11 utilisées par notre algorithme (c'était spécialement le cas pour les fontes Courier). Le tableau B.2 présente la précision atteinte dans cette expérience. Compte tenu de l'utilisation d'un alphabet de près de 200 signes, et de l'absence totale d'informations lexicales, les résultats nous semblent satisfaisants. En particulier, l'abondance de taux de précision supérieurs à 99.9% est très réjouissante.

Comme le montre la figure B.2, la qualité des résultats augmente régulièrement entre 8pt et 14pt, puisqu'il

Fonte	Précision					
	8 pt	9 pt	10 pt	11 pt	12 pt	14 pt
ag-r-r	0.968	0.968	0.968	0.966	0.966	0.965
ag-r-i	0.927	0.901	0.911	0.944	0.960	0.952
ag-b-r	0.812	0.960	0.940	0.964	0.965	0.951
ag-b-i	0.920	0.920	0.957	0.962	0.963	0.960
cr-r-r	0.995	0.982	0.850	0.997	0.994	0.954
cr-r-i	0.994	0.984	1.000	0.992	0.971	0.963
cr-b-r	0.998	0.987	0.992	0.997	0.993	0.976
cr-b-i	0.851	0.995	0.844	1.000	0.997	0.987
hv-r-r	0.961	0.968	0.958	0.974	0.970	0.971
hv-r-i	0.960	0.931	0.960	0.971	0.972	0.976
hv-b-r	0.880	0.968	0.968	0.966	0.965	0.965
hv-b-i	0.931	0.968	0.967	0.966	0.965	0.965
lb-r-r	1.000	0.977	0.994	1.000	1.000	0.996
lb-r-i	0.954	0.948	0.992	0.994	0.999	0.999
lb-b-r	0.991	0.964	1.000	1.000	1.000	1.000
lb-b-i	0.946	0.972	0.978	1.000	1.000	1.000
nc-r-r	0.982	0.996	0.998	0.995	0.998	0.995
nc-r-i	0.945	0.799	0.994	0.998	1.000	0.989
nc-b-r	0.988	0.992	0.999	0.998	1.000	0.998
nc-b-i	0.970	0.968	0.991	0.999	1.000	0.993
pl-r-r	0.996	0.992	0.974	0.997	0.985	0.964
pl-r-i	0.985	0.988	0.996	0.994	0.996	0.994
pl-b-r	0.992	1.000	0.999	0.999	0.994	0.994
pl-b-i	0.985	0.975	0.984	0.995	0.986	0.989
tm-r-r	0.977	0.980	0.995	0.997	0.976	0.986
tm-r-i	0.850	0.971	0.974	0.959	0.978	0.993
tm-b-r	0.950	0.982	0.992	1.000	0.984	0.999
tm-b-i	0.959	0.955	0.972	0.969	0.945	0.990
zc-r-r	0.884	0.978	0.997	0.990	0.987	0.987

Tableau B.2 : Précision de l'OCR monofonte sur les documents LaTeX scannés.

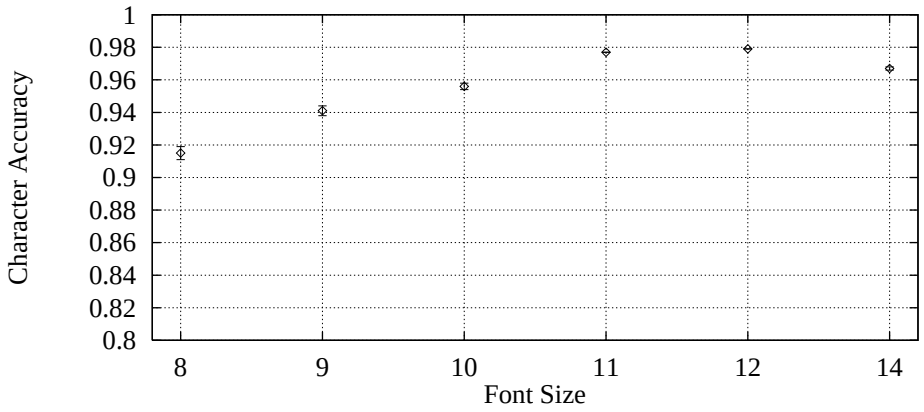


Figure B.2 : Influence de la taille sur la précision.

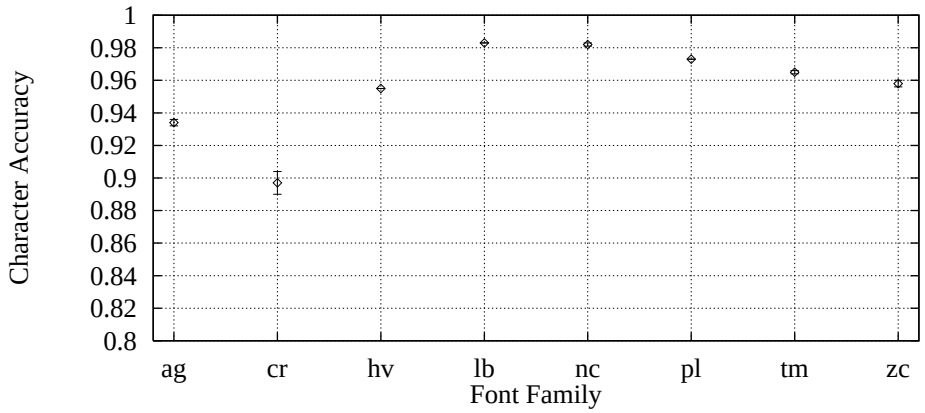


Figure B.3 : Influence de la famille sur la précision.

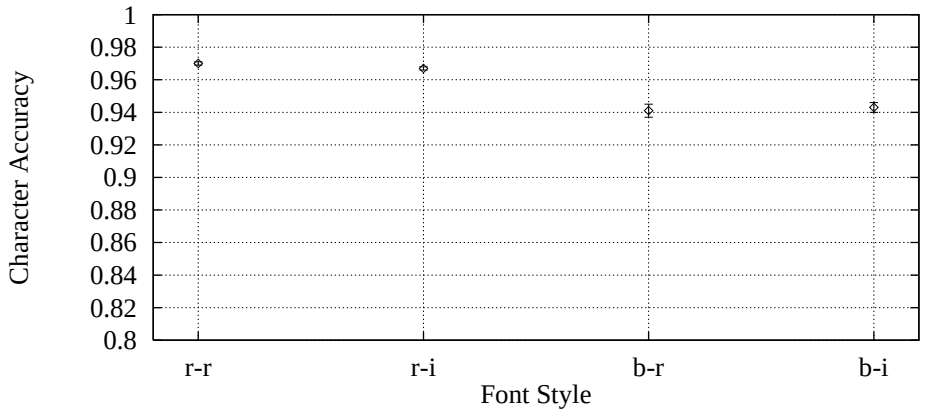


Figure B.4 : Influence du style sur la précision.

y a plus d'information significative. L'apparente exception de la taille 14pt provient d'un artefact, dû à un sous-ensemble de fontes qui diffèrent passablement entre X11 et LaTeX.

La figure B.3 indique qu'il y a quelques variations de qualité entre polices, toutes tailles et tous styles confondus. Le mauvais score pour la police Courier est uniquement dû aux fontes Courier-Bold utilisées par LaTeX, qui ne ressemblent pas à notre version sous X11. Les jambages sont près de deux fois plus longs sous X11, et notre OCR commet beaucoup d'erreurs sur les mots contenant les lettres «jgqy». Les deux polices sans sérif (AvantGarde et Helvetica) sont moins bien reconnues, en raison des confusions entre 'l' minuscule et 'I' majuscule (cf. section 8.2).

La figure B.4 montre que la méthode est peu sensible au style. Ceci contraste avec les performances des OCR actuels, qui sont nettement moins bons avec l'italique.

L'influence du niveau de dégradation des images sur la précision des résultats est illustrée sur la figure B.5. Comme prévu, le taux de reconnaissance diminue lorsque le bruit artificiel augmente.

Le tableau B.3 rapporte une expérience portant sur les documents X11 scannés, de fontes 10pt, avec un alphabet réorganisé pour réduire les confusions entre `l` et `I`. Les résultats sont analogues à ceux présentés

Fonte		Fonte		Fonte		Fonte	
ag-r-r-10	0.992	nc-r-r-10	1.000	lb-r-r-10	1.000	tm-r-r-10	0.999
ag-r-i-10	0.997	nc-r-i-10	0.999	lb-r-i-10	1.000	tm-r-i-10	0.999
ag-b-r-10	0.948	nc-b-r-10	1.000	lb-b-r-10	1.000	tm-b-r-10	1.000
ag-b-i-10	0.992	nc-b-i-10	0.999	lb-b-i-10	1.000	tm-b-i-10	1.000
cr-r-r-10	0.998	pl-r-r-10	1.000	hv-r-r-10	0.996	zc-r-r-10	0.998
cr-r-i-10	0.999	pl-r-i-10	1.000	hv-r-i-10	0.991		
cr-b-r-10	0.998	pl-b-r-10	1.000	hv-b-r-10	0.999		
cr-b-i-10	0.998	pl-b-i-10	1.000	hv-b-i-10	0.996		

Tableau B.3 : Précision de l'OCR monofonte sur les documents X11 scannés.

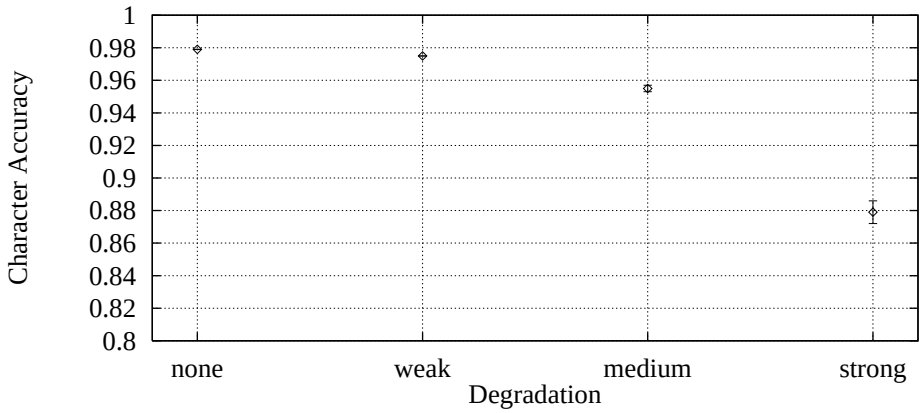


Figure B.5 : Influence de la dégradation sur la précision.

dans la section 8.2, qui concernent les versions synthétiques et dégradées artificiellement. Globalement, il est apparu que les expériences où les paramètres sont communs à toutes les fontes ne sont pas très parlantes : il y a toujours quelques fontes pour lesquelles un autre paramétrage donnerait un bien meilleur résultat (commen la fonte `ag-b-r-10` dans le tableau B.3). Toutefois, ce phénomène ne constitue pas vraiment une limite de l'approche, car notre mesure de dissemblance reste consistante par rapport à la paramétrisation. En clair, cela signifie que notre analyseur serait en mesure de choisir les meilleurs paramètres de façon automatique. C'est par exemple le cas pour le degré d'érosion/dilatation, qui pourrait être choisi en fonction de l'épaisseur moyenne du tracé dans toute la fonte, ou mieux être différencié pour chaque caractère. Notre analyseur sera prochainement étendu avec des mécanismes qui serviront à minimiser l'effort de configuration, ou à autoriser une forme d'apprentissage incrémental.

Annexe C

Simulation de lentilles magiques en Tcl-Tk

Cette annexe présente un mécanisme rudimentaire qui permet de simuler en Tcl-Tk le concept de *lentille magique*, évoqué dans la section 3.1.3. Nous avons vu que ce paradigme se révèle très utile pour gérer une superposition d'informations, en particulier dans un environnement de reconnaissance de documents assistée.

A notre connaissance, l'émulation de lentilles magiques n'a pas été discutée dans la communauté Tcl-Tk, bien que le concept a maintenant acquis une certaine notoriété dans le domaine des interfaces graphiques. Seul le paquetage non-standard Pad++ [166] propose une solution pour gérer plusieurs formes de widgets «transparents», permettant notamment d'exprimer la fonctionnalité des lentilles magiques. Notre objectif n'est pas de faire le tour complet de la question, mais simplement de proposer une solution pragmatique.

C.1 Problématique et objectifs

Une lentille magique est un élément graphique que l'on superpose à des informations affichées, de manière à en révéler une autre vue. Les informations présentées et la nature de la transformation opérée par la lentille sont naturellement dépendantes de l'application. Par exemple, une image de document pourrait supporter une lentille qui offre un facteur de grossissement, ou une autre qui positionne les résultats de l'OCR.

Idéalement, la lentille devrait pouvoir être manipulée sur tout l'écran. Partout où la transformation de vues n'a pas de sens, la lentille devrait fonctionner comme une fenêtre transparente. Une telle fonctionnalité implique de travailler au niveau du gestionnaire de fenêtrage, ce qui peut être assez complexe.

Pour notre part et pour simplifier le codage, nous avons choisi de limiter le contexte d'utilisation d'une lentille. Nous nous sommes fixés le cahier des charges suivant :

- la solution doit être écrite entièrement en Tcl-Tk, sans utiliser d'extension spéciale ni inclure du code C;
- le mécanisme doit s'exprimer avec suffisamment de simplicité;
- lors du déplacement de la lentille, la mise à jour doit sembler quasi-instantanée.

C.2 Solution proposée

Notre solution utilise le widget standard qu'offre Tcl-Tk pour gérer des dessins, à savoir le *canvas*. Le principe est le suivant. Chaque vue est représentée par un canvas distinct. L'une de ces vues correspond à l'affichage normal. Une lentille magique sur cette vue est simulée au moyen d'un élément de dessin (canvas item) de type fenêtre, qui est attaché au canvas correspondant.

La figure C.1 présente la spécification de notre paquetage pour le support de lentilles magiques en Tcl-Tk. La vue normale est représentée par un certain canvas *hôte*. Une lentille magique sur la vue normale est définie par un canvas *cible* qui contiendra la représentation d'une autre facette du contenu du canvas hôte. Afficher la lentille, c'est dessiner dans le canvas hôte une lucarne vers le canvas cible, en faisant coïncider les coordonnées au centre de la lucarne rectangulaire.

Notre spécification permet de modifier certaines propriétés d'une lentille, comme la dimension de la lucarne, et le facteur d'agrandissement entre les systèmes de coordonnées des deux canvas. Trois routines permettent d'activer, de désactiver, et de déplacer la lentille. Une autre routine offre une utilisation encore simplifiée,

```
proc lens_create        { lens_name host_canvas target_canvas }
proc lens_destroy       { lens_name }
proc lens_standard_bind { lens_name }
proc lens_show          { lens_name }
proc lens_hide          { lens_name }
proc lens_move          { lens_name x y }
proc lens_resize        { lens_name w h }
proc lens_set_factor    { lens_name factorx factory }
```

Figure C.1 : Spécification du module de simulation des lentilles magiques.

```
proc demo_lens {} {
  canvas .c1 -width 500 -height 500 -scrollregion {0 0 1500 500}
  canvas .c2 -background yellow
  canvas .c3 -background green
  set f 2.0

  set items ""
  lappend items [.c1 create text 320 20 -text "coucou"]
  lappend items [.c1 create rect 350 100 380 120]
  foreach i $items {
    scan [.c1 coords $i] "%f %f" x y
    .c2 create text $x $y  -fill red -text $i -anchor c
    scan [.c1 bbox $i] "%f %f %f %f" x1 y1 x2 y2
    .c3 create rect $x1 $y1 $x2 $y2
  }
  .c3 scale all 0 0 $f $f
  lens_create my_idnb_lens .c1 .c2
  lens_create my_bbox_lens .c1 .c3
  lens_set_factor my_bbox_lens $f $f
  lens_standard_bind my_idnb_lens
  bind .c1 <a> "lens_resize my_idnb_lens 20 70"
  bind .c1 <b> "lens_resize my_idnb_lens 200 80"
  bind .c1 <c> "lens_standard_bind my_idnb_lens"
  bind .c1 <d> "lens_standard_bind my_bbox_lens"
  pack .c1; focus .c1
}
```

Figure C.2 : Exemple d'utlisation de nos lentilles magiques.

en codant le comportement standard suivant : la lentille est activée lorsque le bouton gauche de la souris est pressé en combinaison avec la touche majuscule, et elle suit ensuite le curseur dans ses déplacements.

La détermination des informations à afficher dans chaque vue est entièrement sous la responsabilité de l'utilisateur.

C.3 Exemple

La figure C.2 présente un script qui illustre l'utilisation de notre module de lentille magique. On dessine dans le canvas hôte quelques éléments graphiques (du texte et un rectangle). Puis le programme prépare deux autres vues qui seront associées chacune à une lentille. La première contient les numéros d'identification de chaque élément graphique de l'hôte. La deuxième vue dessine un rectangle englobant autour de chaque élément graphique, avec un facteur d'agrandissement. Grâce à des commandes associées au clavier, on peut tester le redimensionnement d'une lentille, ou le passage d'une lentille à l'autre.

La figure C.3 illustre l'exécution de notre programme, en montrant l'affichage de la vue normale, et l'apparence de chacune des deux lentilles.

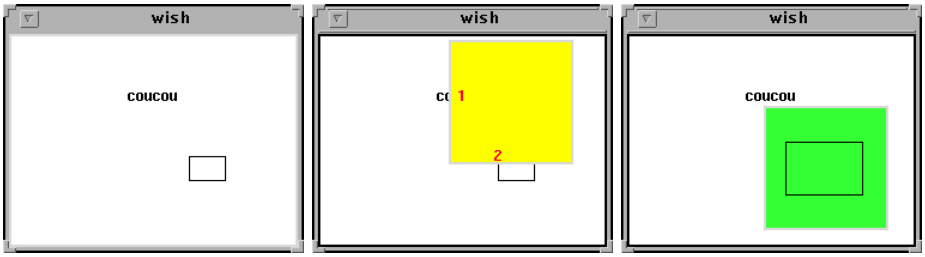


Figure C.3 : Vue normale, puis avec chacune des deux lentilles magiques.

```
global lenses_info
#       lenses_info($lens_name,host)     host canvas, holding the normal view
#       lenses_info($lens_name,target)   target canvas, holding the lens view
#       lenses_info($lens_
#       lenses_info($lens_name,item)     item nb of the lens in host canvas
#       lenses_info($lens_name,factorx)  horizontal scale factor
#       lenses_info($lens_name,factory)  vertical   scale factor
#       lenses_info($lens_name,w)        demi-width  of lens item
#       lenses_info($lens_name,h)        demi-height of lens item
#       lenses_info($lens_name,sw)       width  of host canvas scroll region
#       lenses_info($lens_name,sh)       height of host canvas scroll region

proc lens_create { lens_name host_canvas target_canvas } {
  global lenses_info

  set default_side 100
  set sr [$host_canvas cget -scrollregion]
  $target_canvas configure \
    -width $default_side -height $default_side \
    -scrollregion $sr \
    -borderwidth 0 \
    -confine no
  set i [$host_canvas create window 0 0 -window $target_canvas -anchor c]
  set lenses_info($lens_name,item)    $i
  ...
}
proc lens_move { lens_name x y } {
  ...
  $c coords $i $x $y
  $t xview moveto [expr ($fx*$x-$w)/double($sw)]
  $t yview moveto [expr ($fy*$y-$h)/double($sh)]
}
proc lens_show { lens_name } { ... $c itemconf $i -window $t }
proc lens_hide { lens_name } { ... $c itemconf $i -window "" }
...
```

Figure C.4 : Extrait de l'implémentation du module de lentilles magiques.

C.4 Implémentation

L'implémentation de notre module de lentille magique ne pose pas de difficulté particulière. La figure C.4 présente un extrait de notre réalisation. Un enregistrement de neuf champs est associé à chaque instance de lentille, par l'intermédiaire d'un tableau global. La création d'une lentille suppose que les deux canvas existent déjà; un élément de dessin du type fenêtre est ajouté dans le canvas hôte, et relié au canvas cible. Pour déplacer la lentille, il faut modifier la position de la fenêtre dans le canvas hôte, et faire scroller le canvas cible.

C.5 Critique et extensions

Notre solution possède l'avantage de la simplicité. L'implémentation est très courte. L'exemple donné précédemment montre que quelques lignes suffisent pour ajouter les fonctionnalités d'une lentille magique dans un programme.

Les limites de notre paquetage se situent sur deux plans :

- La transformation de la vue normale par la lentille n'est exprimée qu'indirectement, par une description statique de la nouvelle vue. Notre solution n'aide pas le programmeur à encoder la sémantique sous-jacente. Aucun support n'est offert pour faciliter la gestion du contenu de chaque vue. Par exemple, les éléments graphiques communs aux deux vues sont dupliqués explicitement, et on laisse ouvert le problème de la synchronisation des informations, si le contenu de la vue normale évolue.
- La solution proposée ne permet pas de combiner entre elles plusieurs lentilles. En général, le recouvrement de deux lentilles devrait donner accès à une vue supplémentaire.

Plusieurs parades peuvent être envisagées pour atténuer ces points faibles. Par exemple, on pourrait définir un utilitaire qui recopie régulièrement, de l'hôte vers la cible, tous les éléments graphiques communs. Plus généralement, on peut définir dans une procédure la règle de transformation d'un élément graphique. Le système se chargerait ensuite, à intervalles réguliers, de rafraîchir la vue de la lentille en appliquant cette règle sur chaque élément graphique, en tout cas ceux qui sont concernés par la région de la lentille.

Le problème de la combinaison de lentilles entre elles est plus délicat. Une idée consiste à mettre en oeuvre un autre niveau de lentilles, mais cette fois dans le canvas cible. Si la faisabilité de l'approche doit pouvoir être démontrée, nous ne sommes pas convaincus que la solution serait utilisable en pratique. La gestion des informations graphiques risquerait de devenir un casse-tête pour trois lentilles et plus.

Malgré ses limites actuelles, notre module a pu être utilisé avec succès dans le prototype de reconnaissance de structure physique.