HAL
open science

ODILE : un outil d'intégration extensible de dictionnaire et lemmatiseurs

Isabelle Tomasino

To cite this version:

Isabelle Tomasino. ODILE : un outil d'intégration extensible de dictionnaire et lemmatiseurs. Interface homme-machine [cs.HC]. 1990. dumas-00338014

HAL Id: dumas-00338014

https://dumas.ccsd.cnrs.fr/dumas-00338014

Submitted on 10 Nov 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE DE GRENOBLE (C.U.E.F.A.)

MEMOIRE

présenté en vue d'obtenir

LE DIPLOME D'INGENIEUR C.N.A.M.

en

INFORMATIQUE

par

Isabelle TOMASINO

ODILE : un Outil d'Intégration Extensible de Dictionnaires et de Lemmatiseurs

---

Les travaux relatifs à ce mémoire ont été effectués au Groupe d'Etude pour la Traduction Automatique (UJF&CNRS) sous la direction de Monsieur Christian BOITET.

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE DE GRENOBLE (C.U.E.F.A.)

MEMOIRE

présenté en vue d'obtenir

LE DIPLOME D'INGENIEUR C.N.A.M.

en

INFORMATIQUE

par

Isabelle TOMASINO

ODILE : un Outil d'Intégration Extensible de Dictionnaires et de Lemmatiseurs

---

Les travaux relatifs à ce mémoire ont été effectués au Groupe d'Etude pour la Traduction Automatique (UJF&CNRS) sous la direction de Monsieur Christian BOITET.

Je tiens à remercier,

Monsieur Claude KAISER, professeur au Conservatoire National des Arts et Métiers, président du jury,

Monsieur François PECCOUD, directeur du Groupe d'Etudes pour la Traduction Automatique, pour m'avoir accueillie dans son laboratoire et pour sa présence en tant que membre du jury,

Monsieur Christian BOITET, co-directeur du GETA, pour m'avoir proposé ce sujet, pour ses nombreux conseils et pour l'intérêt constant qu'il a porté à mon travail,

Monsieur Jacques COURTIN, professeur à l'Université des Sciences Sociales de Grenoble, responsable de l'équipe TRILAN du Laboratoire de Génie Informatique et membre du jury, qui m'a permis d'entrer en contact avec le GETA et d'y préparer ce mémoire,

Monsieur Kamel GADDAS, chef de produits de la société WinSoft, pour sa collaboration technique, ses encouragements et sa patience, et aussi pour avoir accepté de participer à ce jury.

Je voudrais également remercier,

les membres de l'équipe TRILAN du Laboratoire de Génie Informatique qui m'ont toujours chaleureusement accueillie dans leurs murs, et en particulier Damien GENTHIAL, pour sa disponibilité,

Joelle COUTAZ et Laurence NIGAY pour m'avoir consacré un peu de leur temps,

Hervé BLANCHON, qui prépare une thèse au GETA, pour l'aide qu'il m'a apporté au niveau de l'apprentissage du Macintosh, et Mathieu LAFOURCADE, en DEA d'informatique, pour sa coopération.

Je remercie également l'ensemble du personnel du GETA, qui, par son accueil sympathique, m'a permis de réaliser ce travail dans de bonnes conditions.

Je remercie enfin la société ELF FRANCE qui, en autorisant un congé-formation, m'a permis de me consacrer pendant une année à la préparation de ce mémoire.

Résumé

Ce mémoire traite de la conception et de la réalisation d'un logiciel interactif d'assistance linguistique sur Macintosh. Il doit permettre la mise en relation de textes en langue naturelle avec des dictionnaires usuels.

Pour cela, le logiciel à concevoir doit intégrer et faire coopérer des outils préexistants, à savoir un lemmatiseur, chargé de la fabrication d'entrées potentielles de dictionnaire après analyse du texte en entrée, et un outil d'accès à des dictionnaires. Il doit être le plus universel possible pour pouvoir exploiter la diversité des lemmatiseurs et des outils dictionnaires existants. Les lemmatiseurs PILAF et ATEF et différents outils d'accès aux dictionnaires tels que WinTool™, WinFile™ et Word Finder® sont présentés, avant de fixer notre choix sur PILAF et WinTool™ pour la réalisation de notre prototype. Dans ce mémoire, nous proposons, pour l'intégrateur ODILE, une interface utilisateur qui réponde à un certain nombre de principes ergonomiques. Nous définissons également des interfaces programme générales pour la communication des deux logiciels avec notre outil.

Nous nous sommes également préoccupé de l'évolution de notre outil ; en particulier nous souhaitions en faciliter la maintenance. Le souhait d'une architecture modulaire pour ODILE a été l'occasion, pour nous, d'étudier le modèle PAC de structuration logicielle des systèmes interactifs. Dans l'implémentation actuelle, nous en avons retenu certains aspects intéressants .

Le besoin d'améliorer les performances du lemmatiseur intégré nous a conduit à réviser la structure de son dictionnaire. Nous examinons plusieurs organisations possibles.

TABLE DES MATIERES

Table des matières 1

Introduction 5

Chapitre I - Traitement de textes et outils linguistiques.

1. Vue générale et contexte 9
1.1. Le projet 9
1.2. Découpe fonctionnelle 10
1.2.1. Le lemmatiseur 10
1.2.1.1. Trois notions de base 11
1.2.1.1.1. Occurrence 11
1.2.1.1.2. Segment 11
1.2.1.1.3. Lemme 11
1.2.1.2. L'analyse morphologique 12
1.2.1.3. La génération de lemmes 13
1.2.2. L'outil d'accès aux dictionnaires 14
1.2.3. Le module chef d'orchestre 15
1.3. Logiciels exploitables 15
1.3.1. Logiciels de lemmatisation 16
1.3.1.1. PILAF 16
1.3.1.2. ATEF 20
1.3.1.3. Autres analyseurs 21
1.3.2. Logiciels d'accès à des dictionnaires 23
1.3.2.1. WinTool™ 23
1.3.2.2. WinFile™ 23
1.3.2.3. Word Finder® 23
1.3.3. Orientation 24
2. Objectifs et contraintes 25
2.1. Objectifs 25
2.1.1. ODILE 25
2.1.1.1. Ergonomie de l'outil, structuration du système interactif 25
2.1.1.2. "Propreté" de l'intégrateur 26

2.1.1.3. Généricité de l'outil....26
2.1.2. Le lemmatiseur....27
2.1.2.1. Structure des dictionnaires de PILAF....27
2.1.2.2. Les contraintes de lemmatisation....28
2.1.3. L'outil d'accès au dictionnaire....28
2.2. Contraintes....29
2.2.1. Aspects matériels....29
2.2.2. Aspects logiciels....29
2.2.3. Aspects temporels....30
2.3. Choix....30

Chapitre II - Conception.

1. Introduction....36
1.1. Problèmes rencontrés....36
1.1.1. Problèmes liés à la forme de l'outil....36
1.1.2. Problèmes liés à l'intégration de modules....37
2. L'interface homme-machine....39
2.1. Spécifications abstraites....39
2.1.1. Principe de génération de clés de recherche....39
2.2.2. Les trois classes de dictionnaires....42
2.2. Ergonomie....43
2.2.1. Spécifications externes d'ODILE....43
2.2.1.1. Fonctionnalités d'ODILE....43
2.2.1.2. Détail du menu ODILE....47
2.2.2. Principes pratiques ergonomiques....58
2.3. Le modèle PAC et son application dans ODILE....63
2.3.1. Le modèle PAC....63
2.3.1.1. Définition....63
2.3.1.2. Intérêt du modèle PAC....65
2.3.2. Architecture du système interactif dans ODILE....66
3. La communication dans ODILE....71
3.1. Interface avec les lemmatiseurs....72
3.1.1. Fonctionnalités des lemmatiseurs....72
3.1.2. Choix de la structure d'entrée des lemmatiseurs....73
3.1.3. Choix de la structure de sortie des lemmatiseurs....74
3.1.3.1. Choix de la structure....74
3.1.3.2. Informations fournies....77
3.1.4. Points d'entrée des lemmatiseurs....81

3.1.5. Interfaçage lemmatiseurs - ODILE 81
3.1.5.1. Paramètres d'entrée des lemmatiseurs 81
3.1.5.2. Paramètres de sortie des lemmatiseurs 83
3.2. Interface avec les outils dictionnaire 83
3.2.1. Fonctionnalités des outils dictionnaire 83
3.2.2. Points d'entrée de WinTool™ 84
3.2.3. Interfaçage outils dictionnaire - ODILE 85
4. Structure des dictionnaires 86
4.1. Quelques exemples d'organisation de fichiers 87
4.1.1. Dictionnaires WinTool™ et organisation en B-arbres 87
4.1.2. Dictionnaire de clés personnelles 88
4.1.3. Dictionnaires de lemmatisation 89
4.1.3.1. Le dictionnaire PILAF 89
4.1.3.2. Le dictionnaire ATEF 94
4.2. Solution proposée 95

Chapitre III - Réalisation.

1. Le contexte et l'apprentissage 103
1.1. Logiciels 103
1.2. Système 103
1.2.1. Interface utilisateur et programmation par événements 104
1.2.1.1. Interface utilisateur 104
1.2.1.2. Logiciels pilotés par événements 104
1.2.2. Gestion de la mémoire 106
1.3. Méthodologie retenue 108
2. Contenu de la première version 108
2.1. Adaptation des logiciels 108
2.1.1. Adaptation de PILAF 109
2.1.1.1. Version brute 109
2.1.1.1.1. Adaptation à THINK Pascal™ 109
2.1.1.1.2. Adaptation au Macintosh 111
2.1.1.1.3. Portage des fichiers de données 117
2.1.1.2. Version intégrable à ODILE 119
2.1.1.2.1. Adaptation du code 119
2.1.1.2.2. Normalisation des noms d'unités et de fichiers 120
2.1.2. Adaptation de WinTool™ 120
2.2. Fonctions propres d'ODILE 121
2.3. Interface utilisateur 124

2.4. Architecture logicielle..... 125
3. Perspectives et bilan..... 130
3.1. Amélioration des performances d'ODILE..... 130
3.2. Travail restant sur la première version..... 131
3.3. Extensions envisageables..... 132

Conclusion..... 135

Annexes

Annexe 1 . Fichier des contraintes et fichier des classes lexicales de PILAF sous format externe standard..... 141
Annexe 2 . Exemple de fichier texte résultat d'un lemmatiseur intégrable à ODILE..... 147

Bibliographie..... 149

INTRODUCTION

Grâce au formidable essor de la bureautique au cours des dix dernières années, les micro-ordinateurs ont envahi tous les domaines d'activité. La documentation, manuscrite à l'origine, s'est radicalement transformée à l'heure actuelle pour céder le place à la documentation électronique, et cela notamment grâce aux logiciels de traitement de texte.

Le foisonnement de tels outils sur le marché, pour tous types de micro-ordinateurs, laisserait à penser que les rédacteurs sont, aujourd'hui, des utilisateurs comblés. Or, il n'en est rien.

Dans de nombreux cas, l'utilisateur souhaiterait, en cours de travail, pouvoir accéder à des bases documentaires, dans un but d'aide à la rédaction ou d'aide à la traduction par exemple. Quelques logiciels tentent d'apporter une solution à ce problème. Nous pouvons évoquer ici WordFinder® de la société Microlytics et WinTool™ de la société WinSoft.

Ces deux outils sont des accessoires de bureau sur Macintosh™. Ils permettent entre autres, et c'est ce qui nous intéresse ici, de rechercher un élément dans une base documentaire à partir de sa clé qui est, soit une sélection dans le texte de l'application courante, soit le résultat d'une saisie manuelle. Une contrainte énorme de ce type d'outils réside dans le fait que, pour être trouvé, l'élément recherché dans la base documentaire doit être une clé dans cette base. Nous pouvons regretter également que ces logiciels soient généralement restrictifs quant aux bases documentaires accessibles.

Le GETA (Groupe d'Etude pour la Traduction Automatique) a donc eu l'idée de concevoir un outil répondant mieux aux attentes des utilisateurs en exploitant ses compétences en matière de traitement automatique du langage naturel.

Le logiciel résultant ne sera pas seulement un logiciel de plus. L'idée est de construire un logiciel, baptisé ODILE, qui soit un véritable Outil d'Intégration Extensible de Dictionnaires et de Lemmatiseurs, et fonctionne en accessoire de bureau, de façon à être utilisable depuis n'importe quelle application.

Le premier chapitre de ce mémoire est l'occasion, par un exposé des objectifs à atteindre et des contraintes à satisfaire, de mettre en évidence les caractéristiques qui font l'originalité d'ODILE.

Le second chapitre met l'accent sur trois problèmes auxquels nous avons été confrontés au cours de la conception et qu'il nous a paru intéressant de présenter dans ce mémoire. Nous abordons en premier lieu la conception de l'interface homme-machine et nous voyons l'influence de l'ergonomie et du choix de l'architecture d'un système interactif sur la qualité globale du système final. Nous étudions dans un deuxième temps les problèmes liés à l'intégration de logiciels préexistants. Enfin, nous discutons, dans la troisième section, la possibilité d'améliorer l'organisation des dictionnaires du lemmatiseur PILAF, développé par l'équipe TRILAN du Laboratoire de Génie Informatique et intégré à la première version d'ODILE, cela en vue d'améliorer les performances globales du système.

Dans le troisième chapitre, nous présentons nos solutions aux problèmes présentés dans le chapitre précédent, la méthodologie de mise en œuvre de ces solutions, les difficultés rencontrées.

Le bilan global sur l'avancement du projet est donné dans une dernière partie. Nous concluons par des remarques sur les améliorations, les extensions possibles et l'apport personnel de cette étude.

Chapitre I

Traitement de textes et outils linguistiques

1. Vue générale et contexte

1.1. Le projet

Le succès actuel de la bureautique encourage les développeurs à proposer des systèmes intelligents de gestion de consoles, particulièrement en ce qui concerne les éditeurs de texte.

Actuellement, un véritable logiciel de traitement de texte ne doit plus seulement proposer des services de mise en forme de texte. De plus en plus nombreux sont ceux qui reconnaissent qu'il faut apporter aux traitements de texte des services d'assistance linguistique et des aides documentaires.

Les textes à traiter étant exprimés en langage naturel, il est possible de coupler les commodités existantes, à savoir les manipulations variées de caractères, de lignes, de paragraphes, etc., à des programmes morpho-lexicaux, à des dictionnaires, à des analyseurs syntaxiques rudimentaires, pour augmenter la capacité des éditeurs de texte.

La correction orthographique est un de ces services et fait l'objet de nombreuses études menées par différentes équipes en Europe et notamment en France.

Quant à nous, nous nous intéressons, dans ce projet, aux aides documentaires. Notre but est de permettre à un utilisateur qui manipule du texte, d'accéder à de la documentation, de quelque nature qu'elle soit, pour fournir une véritable assistance linguistique, dans un but d'aide à la rédaction, d'aide à la traduction, etc.

De façon concrète, à partir d'une sélection dans un texte visualisé sous une application, l'utilisateur doit pouvoir accéder directement à la documentation correspondante. Il est à noter que nous désirons pouvoir effectuer cela depuis toute application traitant du texte, donc depuis pratiquement toutes les applications : les traitements de texte, bien sûr, mais aussi les systèmes de gestion de bases de données, les tableurs, certains outils graphiques (comme MacDraw), et les hypertextes.

Prenons un exemple. Supposons que l'utilisateur souhaite trouver une traduction de la forme "chevaux de bataille" qui figure dans son texte. Nous supposons aussi qu'il dispose d'un dictionnaire bilingue en ligne.
Avec un outil classique d'accès à des dictionnaires, l'utilisateur doit fournir lui-même la clé d'accès correspondante "cheval de bataille".
Avec ODILE, il n'aura qu'à sélectionner avec la souris la forme "chevaux de bataille" dans son texte ; ODILE se chargera de générer automatiquement "cheval de bataille" et s'en servira pour accéder au dictionnaire et donner une traduction à l'utilisateur.

Cet exemple simple met déjà en évidence le fait que notre outil doit intégrer des fonctionnalités diverses et complémentaires.

1.2. Découpe fonctionnelle

Si nous poussons un peu plus loin l'analyse, l'outil à concevoir se compose de trois parties principales : le module de lemmatisation, le module d'accès aux dictionnaires, et le module que nous appelons "chef d'orchestre", chargé de faire communiquer les deux premiers modules et d'organiser les traitements.

1.2.1. Le lemmatiseur

Qu'est-ce qu'un lemmatiseur?

En première approximation, c'est un outil permettant de faire l'analyse d'une suite d'occurrences données et de fournir une ou plusieurs suites de lemmes correspondant aux occurrences analysées.

Exemple :
Soit la suite d'occurrences :
"chevaux de frise",
alors le lemmatiseur fournit les suites de lemmes suivantes:
"cheval de frise",
"cheval de friser".
La suite de lemmes "cheval de friser" n'a pas de sens. Elle constitue cependant un des résultats du lemmatiseur dans le cas où celui-ci n'effectue pas d'analyse de contexte.

Nous allons examiner les deux traitements constituant le processus de lemmatisation, à savoir l'analyse morphologique et la génération de lemmes.

Mais précisons d'abord plusieurs termes employés par la suite.

1.2.1.1. Trois notions de base

1.2.1.1.1. Occurrence

Une occurrence est une forme d'un mot dans un texte, c'est-à-dire une suite de caractères encadrés par deux "blancs". Dans la suite de l'exposé, nous utiliserons indifféremment les termes "occurrence" ou "forme".

Exemple :
Dans le texte `"les enfants font du patin à roulettes"`,
`"les"`, `"enfants"`, `"font"`, `"du"`, `"patin"`, `"à"` et `"roulettes"` sont sept occurrences (ou formes) distinctes.

1.2.1.1.2. Segment

Un segment est un morceau d'occurrence. C'est l'entité morphologique élémentaire. Il existe plusieurs catégories de segments, telles que bases et affixes (préfixes, suffixes et désinences). Nous pouvons préciser que la base est la plus longue chaîne de caractères commune à toutes les occurrences d'un mot, ou à un sous-ensemble d'occurrences.

Exemple :
Soit l'occurrence : `"retrouvera"`.
Nous pouvons obtenir le découpage en segments suivant:

préfixe : `"re"`
base : `"trouv"`
suffixe : `"er"`
désinence : `"a"`

1.2.1.1.3. Lemme

La définition du lemme nécessite l'introduction préliminaire des notions de classe syntaxique et de paradigme.

Une classe syntaxique est un sous-ensemble de mots ayant en commun :

- une valeur catégorielle abstraite,
- des catégories grammaticales propres telles que genre et nombre pour les noms, temps et mode pour les verbes.

Un paradigme est un ensemble d'affixes acceptés par une base pour produire des occurrences correctes d'une classe syntaxique.

Un lemme est une famille d'occurrences appartenant à la même classe syntaxique et générées par le même paradigme à partir d'une racine généralement commune. Le lemme est noté par une occurrence canonique.

Dans la suite de l'exposé, nous parlerons souvent de lemme d'une occurrence ; il s'agira de l'occurrence canonique que nous venons d'évoquer et qui correspond généralement à une entrée dans un dictionnaire usuel.

Exemple :

occurrence	"fermes"	
lemmes	"ferme"	nom
	"ferme"	adjectif
	"fermer"	verbe

Nous parlerons aussi de "classe lexicale". Les linguistes ne sont pas toujours d'accord sur la signification des termes. Il s'agit pour nous d'une notion équivalente à celle de "classe syntaxique" désignée également sous le terme de "classe morpho-syntaxique".

1.2.1.2. L'analyse morphologique

Pourquoi avoir recours à une analyse morphologique?

Comme l'expliquent D. COULON et D. KAYSER dans [COUL86], cette opération évite d'avoir à mentionner dans un lexique toutes les formes que peut prendre un mot. On ne conserve que certains morceaux, jugés représentatifs, et on doit retrouver toutes les formes du mot par des règles décrivant les flexions possibles. Le lexique est alors beaucoup moins volumineux ; il est aussi plus rapide et moins fastidieux à créer.

L'analyse morphologique consiste à déterminer des renseignements sur des occurrences constitutives d'une chaîne d'entrée. Il y à plusieurs façons de procéder. L'analyse morphologique décrite dans [COUR77] n'effectue pas de prédécoupage initial de la chaîne d'entrée en occurrences. Elle consiste à déterminer les segments M_i constituant la chaîne d'entrée et à établir les renseignements linguistiques sur ces segments, et par-là à identifier les occurrences puisqu'une occurrence correspond à un segment ou à la concaténation de plusieurs des segments déterminés. Cette analyse est réalisée par un moteur paramètré par une grammaire et des dictionnaires sur lesquels nous reviendrons plus loin. Cela induit un comportement

différent du moteur si la grammaire et les dictionnaires qu'il utilise sont différents, donc plus de souplesse.

Le principe de segmentation est de déterminer les M_i par identification dans un dictionnaire et de contrôler la légitimité de la concaténation des divers segments par application de règles de grammaire, puis de répéter ces deux opérations jusqu'à atteindre la fin du groupe d'occurrences.

Le principe de transduction consiste à donner toutes les interprétations de toutes les décompositions des occurrences en segments.

1.2.1.3. La génération de lemmes

La génération de lemmes s'appuie sur les résultats de l'analyse morphologique présentée ci-dessus.

Les méthodes utilisées sont multiples.

Une méthode naturelle est de reconstituer le ou les lemmes de chaque occurrence au cours de l'analyse morphologique.

Une méthode simple est de retrouver les lemmes dans le dictionnaire ; dans ce cas les lemmes sont mémorisés dans le dictionnaire.

Une autre méthode est celle utilisée par le lemmatiseur extrait de l'analyseur PILAF. La génération des lemmes ne commence que lorsque l'analyse morphologique de toutes les occurrences est effectuée.

Elle s'organise selon le schéma de la figure 1.1 [CHAP89b].

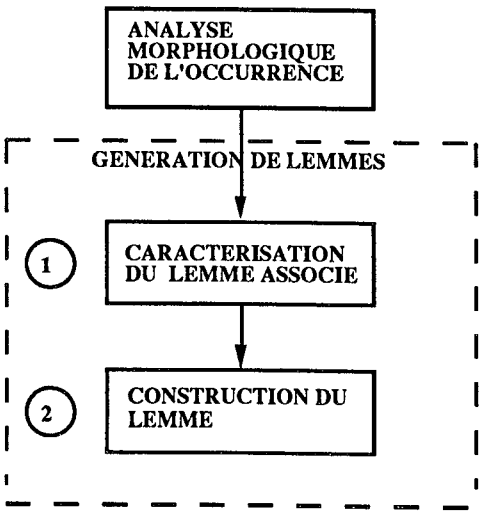


figure 1.1

Pour chacun des résultats de l'analyse morphologique d'une occurrence donnée, l'information "classe lexicale" permet, à l'étape 1 de génération, de définir la classe lexicale et les variables morphologiques du lemme associé.

L'étape 2 consiste, en parcourant les mêmes règles de concaténation de segments que pour l'analyse, à construire une occurrence ayant des caractéristiques morphologiques identiques à celles obtenues à l'étape 1.

Le système fonctionne aussi bien en analyse qu'en génération. En effet, nous pouvons remarquer que les règles de grammaire servent, non seulement à découper la chaîne d'entrée en occurrences par segmentation pour effectuer leur analyse, mais encore à construire des occurrences à partir de segments. La génération est utile entre autres pour tester et valider la grammaire. Nous constatons donc que le système est réversible.

1.2.2. L'outil d'accès aux dictionnaires

Précisons avant tout que le mot "base" apparaîssant dans ce paragraphe fait référence à la terminologie employée généralement dans la documentation décrivant les outils d'accès aux dictionnaires et désigne les fichiers manipulés par ces outils. Ce terme ne désigne donc pas ici le type de segment "base" que nous avons évoqué lors de la présentation de l'analyse morphologique. Nous retrouverons le terme "base" en tant que fichier dans tous les paragraphes traitant des outils d'accès aux dictionnaires.

L'outil d'accès aux dictionnaires doit avoir deux fonctions essentielles :

- il doit permettre l'ouverture d'au moins une base ,
- il doit permettre d'accéder à un élément donné de la base ouverte, en fonction d'une information particulière, sa clé, qui est fournie par l'utilisateur.

Toute autre fonctionnalité que peut proposer ce type d'outil, telle qu'ouverture de plusieurs bases simultanément, gestion des bases (création de bases, modification, suppression d'élément de bases, etc.), constitue un service supplémentaire. C'est en évaluant les avantages de chacun de ces outils que l'utilisateur peut choisir l'outil qui lui convient pour résoudre un problème donné.

Dans la suite de l'exposé, nous préférerons employer le terme de dictionnaires usuels plutôt que le terme de bases souvent employé dans les manuels descriptifs des outils et qui introduit une confusion puisqu'il tend à laisser imaginer qu'il s'agit de bases de données ; or, dans les cas étudiés ici, l'outil d'accès aux dictionnaires consulte et modifie de simples fichiers. De plus, par souci de simplicité, nous préférerons désigner l'outil d'accès aux dictionnaires par l'expression "outil dictionnaire", voire même "dictionnaire" dans certains schémas.

1.2.3. Le module chef d'orchestre

Ce troisième module doit permettre de diriger les traitements, exprimés à travers les commandes de l'utilisateur, vers les modules compétents pour l'exécution de ces traitements. Pour certaines tâches, le module compétent peut d'ailleurs être le module chef d'orchestre lui-même.

Il doit gérer la communication avec l'utilisateur, à travers une interface utilisateur, et également avec les modules de lemmatisation et d'accès aux dictionnaires usuels à travers des interfaces programmes.

Cela est schématisé par la figure 1.2.

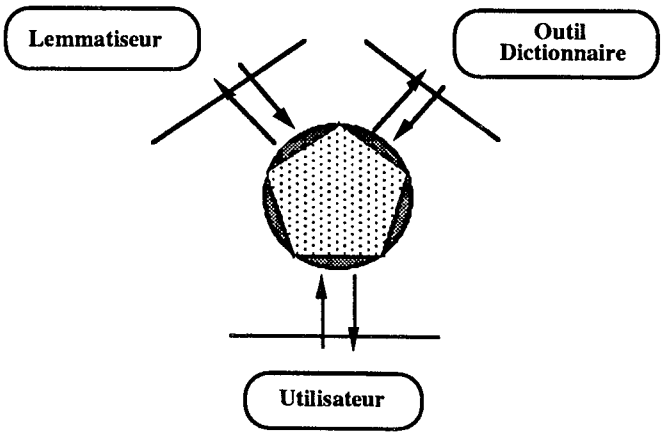


figure 1.2

Pour les raisons mêmes qui justifient l'emploi des techniques du génie logiciel, à savoir maintenance aisée, portabilité accrue des logiciels et plus grande réutilisabilité des modules, nous avons posé en principe qu'en aucun cas l'utilisateur, l'outil de lemmatisation et l'outil dictionnaire ne doivent communiquer directement entre eux.

C'est le module central d'ODILE qui décide des liens à établir pour mener à bien les commandes de l'utilisateur. C'est lui qui organise la coopération harmonieuse des divers composants du logiciel.

1.3. Logiciels exploitables

Un certain nombre de logiciels existants réalisent les fonctions de lemmatisation ou les fonctions d'accès à des dictionnaire usuels. Nous les évoquons ici rapidement.

1.3.1. Logiciels de lemmatisation

1.3.1.1. PILAF

Analyse morphologique

PILAF (Procédures Interactives Linguistiques Appliquées au Français) est, à l'origine, un analyseur morpho-syntaxique développé par le groupe TRILAN (TRaitements Informatiques des LAngues Naturelles) du Laboratoire de Génie Informatique de l'IMAG (Institut de Mathématiques Appliquées de Grenoble) [COUR88].

Nous nous intéressons ici à la mise en oeuvre de l'analyse morphologique dans PILAF et nous abandonnons l'analyse syntaxique qui n'a pas d'intérêt pour la lemmatisation. PILAF est basé sur le modèle du transducteur d'états finis.

Un transducteur est un automate dont l'unité de contrôle peut prendre différents états à l'intérieur d'un ensemble fini. L'état suivant est déterminé par l'état courant, la lecture d'un symbole d'entrée et l'application de la fonction de transition. Le transducteur produit un résultat sur une structure de sortie, chaque fois qu'il exécute une transition.

Dans le cas présent, le vocabulaire d'entrée est déterminé par le dictionnaire, la fonction de transition par la grammaire.

La figure 1.3 présente les trois entités essentielles de l'analyseur morphologique PILAF

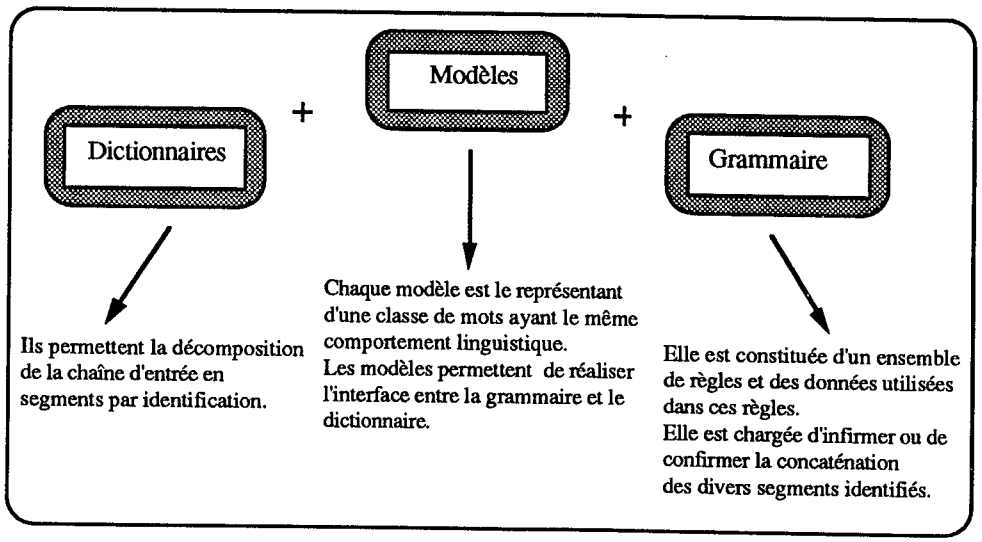


figure 1.3

Les dictionnaires contiennent tous les segments reconnus par l'analyseur PILAF. Leur but est de limiter le nombre d'essais successifs de segmentation.

Il est à noter ici que les dictionnaires sont modifiables de manière incrémentale. PILAF propose deux modes de modification :

- un mode expert où l'utilisateur doit fournir le segment à insérer (base ou terminaison) et le modèle ainsi que d'autres informations que nous détaillerons au paragraphe 4.1.3.2. du deuxième chapitre et qui caractérisent le segment,

- un mode "novice" pour l'insertion de bases par équivalence : l'utilisateur fournit une occurrence contenant le segment à insérer et spécifie que cette occurrence a le même comportement qu'une autre occurrence ayant la même forme.

Exemple :

1- "consigner" comme "manger"
2- "chevaux" comme "journaux"

Dans le cas 1, si l'on suppose que "poser" de base "pos" est le modèle des verbes du premier groupe, la base "consign" de modèle "pos" est insérée dans le dictionnaire.
Dans le cas 2, si "journaux" est le modèle des substantifs en "al", la base "chev" de modèle "journ" est insérée dans la base.

Les modèles autorisent l'application d'une ou plusieurs règles de grammaire. Chaque élément de dictionnaire appartient à une classe représentée par un modèle. Ils sont de la forme :

```
/NOM MODELE/ : REG:= (liste de noms de règles) ou valeur  prédéfinie;
   infos linguistiques;
   VAL:= (liste de noms de règles) ou valeur  prédéfinie;
   SAT:= (liste de noms de règles) ou valeur  prédéfinie.
```

La grammaire est une grammaire à validations et saturations (GVS) équivalente à une grammaire d'état finis mais de formulation plus concise [COUR77]. Les règles sont de la forme :

```
/NOM REGLE/ : infos linguistiques;
   VAL:=(liste de noms de règles) ou valeur  prédéfinie;
   SAT:=(liste de noms de règles) ou valeur  prédéfinie;
   indicateurs.
```

Fonctionnement du transducteur

A l'étape i, le segment s a été identifié dans le dictionnaire.
Si le modèle de s est m (fourni par le dictionnaire), alors la liste de règles applicables à l'étape i est : intersection (REG_m, VAL_i)

Il faut ensuite calculer les nouvelles valeurs de VALidation et de SATuration pour l'étape suivante, soit i+1. Si r est une règle applicable pour le modèle m à l'étape i, alors :

SAT_{i+1} = Union (SAT_i, SAT_r, SAT_m)

VAL_{i+1} = Union (VAL_r, VAL_m) - SAT_{i+1}

A l'initialisation, on a :

SAT_0 = [],

VAL_0 = VAL00 VAL00 est la valeur de la validation initiale fixée par l'utilisateur.

Lemmatisation

Suite aux travaux d'étudiants de DESS-IDC en 1989, le logiciel PILAF a été adapté pour donner naissance à un lemmatiseur constituant un module particulier [CHAP89b]. Ces travaux se situent dans le cadre d'un projet de fin d'études que nous appellerons "projet MacPILAF",

Le module lemmatiseur se compose de l'intégralité de l'analyse morphologique et de traitements s'inspirant de ceux de la génération. Il ne s'agit plus de générer toutes les formes d'une occurrence, mais seulement celles qui répondent à certains critères.

Exemple :

Occurrence	Génération	Lemmatisation
"irons"	"irai"	"aller"
	"iras"	
	"ira"	
	"irons"	
	"irez"	
	"iront"	
	"irais"	
	"irait"	
	"irions"	
	"iriez"	
	"iraient"	
	"ailles"	

```
"aillent"
"aille"
"vas"
"allons"
"allez"
"aller"
"vont"
...
```

Les formes générées sont toutes les formes de la conjugaison du verbe "aller".

Les critères évoqués plus haut sont mémorisés dans une structure effectuant le lien logique entre les modules d'analyse morphologique et de génération: il s'agit du fichier des contraintes de lemmatisation qui fournissent pour chaque classe lexicale de forme, la classe lexicale et les variables morphologiques du lemme associé.

Exemple :

Classe lexicale Forme	Classe lexicale Lemme	Variables Lemme
"participe passé"	"infinitif"	" "
"verbe"	"infinitif"	" "
"substantif commun"	"substantif commun"	"singulier"

...

Pour davantage de détails, on pourra se reporter à l'annexe I où figure la liste d'un fichier texte de contraintes de lemmatisation.

Une structure de données particulière a été mise en place pour la communication entre la phase d'analyse morphologique et la phase de génération adaptée à la lemmatisation. Elle mémorise la ou les bases relatives à chacune des occurrences analysées, ainsi que les informations morphologiques associées. Dans la phase de génération, chaque base est recherchée dans le dictionnaire : on obtient ainsi son modèle. La classe lexicale relative à cette base est recherchée parmi les contraintes de lemmatisation : on obtient la classe lexicale et les variables morphologiques du lemme associé.

Le modèle permet d'activer l'enchaînement des règles applicables pour la génération de formes ; les variables morphologiques du lemme restreignent cette génération à la forme correspondant à ces variables morphologiques.

1.3.1.2. ATEF

ATEF (Analyse de Textes en Etats Finis) est le langage employé pour écrire les analyseurs morphologiques des systèmes de traduction développés sous ARIANE-G5, le générateur de systèmes de TAO (Traduction Assistée par Ordinateur) conçu par le GETA.
Nous le présentons ici car il peut facilement être adapté pour devenir un lemmatiseur.

Les informations que nous présentons ici sont tirées de [CHAU72].

Le système ATEF est fondé sur le modèle de transduction d'états finis. Comme PILAF, ATEF comporte :

- des dictionnaires,
- une grammaire,
- des données caractérisant le comportement des éléments du dictionnaire, équivalents aux modèles de PILAF, les formats.

Le fonctionnement d'ATEF est différent de celui de PILAF. Pour PILAF, le blanc n'est pas un séparateur d'occurrences et celui-ci effectue la segmentation sur la chaîne initiale ramenée à la taille maximum d'un élément du dictionnaire, soit 25 caractères. ATEF, lui, effectue la segmentation sur des formes. Une forme correspond à une suite de caractères comprise entre deux blancs. Les formes sont isolées dans le texte d'entrée par un programme de préédition.

Pour les deux systèmes, la segmentation consiste à décomposer l'unité de traitement en une suite ordonnée de segments par recherche dans des dictionnaires. L'évolution des états est réalisée par application de règles grammaticales. Il est à noter qu'ATEF permet de définir plusieurs lexiques particuliers de tournures.

Cependant, ATEF possède deux mécanismes qui n'existent pas dans PILAF, à savoir une analyse de cohérence et un contrôle de la segmentation. Le premier mécanisme permet de diminuer le nombre de solutions pour une forme et de réduire le nombre de combinaisons entre solutions par des conditions d'application des règles faisant référence jusqu'à quatre formes en arrière. Le second mécanisme permet d'accélérer la segmentation en guidant l'algorithme de découpage, à travers la grammaire. Plus précisément, à un pas donné de la segmentation, les règles applicables peuvent contenir, dans leur partie affectation, des fonctions standard permettant le contrôle de l'algorithme de découpage (fonctions -ARRET-, -ARD-, -ARF-, -STOP-, -FINAL-) ou permettant d'intervenir au niveau des formes (fonctions -TRANS-, -ELIM-).

La structure interne des dictionnaires de PILAF et d'ATEF est différente. En particulier, la modification des dictionnaires ATEF n'est pas possible de manière incrémentale en raison de l'organisation retenue pour ces dictionnaires. Toute modification entraîne une recompilation de ceux-ci. Nous reviendrons sur ce point au chapitre II paragraphe 4.1.3.

1.3.1.3. Autres analyseurs

Nous devons bien sûr évoquer également d'autres classes d'analyseurs morphologiques, parfois appelés analyseurs "sans dictionnaires", par opposition aux analyseurs avec dictionnaires que nous avons présentés plus haut ou du fait du petit nombre de type de données que leurs modèles linguistiques utilisent (listes fermées).

Pour illustrer notre propos, nous présentons l'analyseur morpho-syntaxique du français développé par Patrick PALMER en vue de son application à l'indexation automatique de textes [PALM90]. En fait, seul le traitement morphologique retient notre attention dans le cadre de notre projet. Nous nous limitons donc à l'exposé rapide de cette partie.

Le traitement morphologique consiste à segmenter le texte d'entrée en formes, sans pré-traitement, à partir d'un dictionnaire de racines et d'un ensemble de désinences organisées en tables programmées et compilées en même temps que les algorithmes d'analyse.

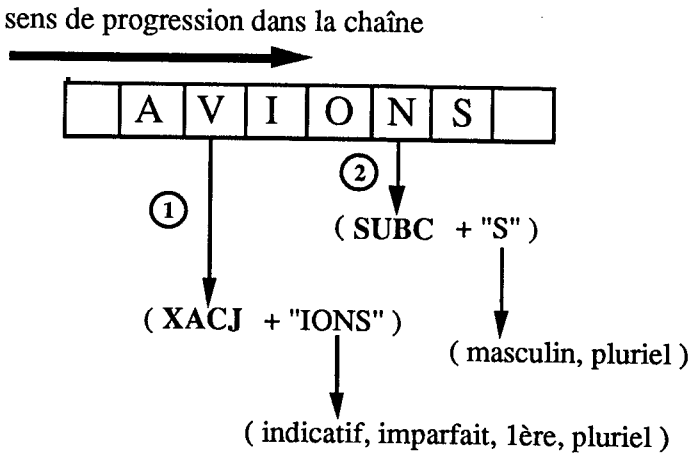
* à chaque racine sont associés, dans le dictionnaire des racines, les renseignements linguistiques suivants :
- une catégorie grammaticale (ex. SUBC, VBCJ, ...) dont le rôle est équivalent à la catégorie lexicale de PILAF,
- un modèle morphologique regroupant un ensemble de désinences juxtaposables à la racine,
- le représentant de l'unité lexicale qui est la forme normalisée (lemme) représentant l'ensemble des occurrences que l'on peut générer à partir de la racine.

* à chaque désinence est associée, dans les tables de désinences, un ensemble de valeurs grammaticales (GENRE, NOMBRE pour un substantif, ...) dont le rôle est équivalent aux variables morphologiques de PILAF.

Une décomposition d'une forme en une racine et une désinence s'effectue d'abord par la reconnaissance, caractère par caractère, de la racine potentielle dans le dictionnaire, puis par l'identification de la désinence parmi l'ensemble des désinences possibles désigné par le modèle associé à la racine reconnue.

Une caractéristique importante de cet analyseur est que, en cours d'analyse, il progresse caractère par caractère sans retour arrière : en une seule passe, il peut déterminer toutes les décompositions possibles d'une forme.

Exemple: tiré de [PALM90], p. 157.



Cela est autorisé par la structure particulière du dictionnaire des racines :

- la factorisation des racines sous forme d'arbre lexicographique permet l'analyse sans "retour arrière" et permet également de minimiser la taille du dictionnaire,
- la priorité donnée aux racines les plus fréquentes (vers le "haut" de l'arbre) assure de plus l'optimisation des accès.

La construction d'un tel arbre allie la technique de construction d'arbre de recherche "Median Split Tree" (MST) à la technique classique de factorisation de chaînes. Pour plus de détails, on pourra consulter [PALM90] p.162 à 164.

Pour information, nous pouvons préciser que la phase d'analyse morphologique de cet analyseur est fortement imbriquée avec des processus d'enrichissement du vocabulaire du dictionnaire et des processus de traitements syntaxiques : le premier processus permet de répertorier dans le dictionnaire des mots jusque là inconnus après que le système a procédé à leur catégorisation automatique ; le deuxième processus permet de confirmer ou d'infirmer la validité de certaines interprétations morphologiques en fonction du contexte.

Il est à noter que, dans le cadre du projet ODILE, cet analyseur est très intéressant. Il permet en effet d'obtenir facilement le lemme correspondant à une occurrence donnée (appelé ici représentant d'unité lexicale), puisqu'il est associé à la racine reconnue dans le dictionnaire.

1.3.2. Logiciels d'accès à des dictionnaires

1.3.2.1. WinTool™

WinTool™ est un accessoire de bureau sur Macintosh développé et commercialisé par la société WinSoft [WINT87]. Il permet de consulter et de modifier des bases documentaires créées par une application différente, DocteurBase®, livrée avec WinTool™.

La structure des bases est très simple. Chaque enregistrement comporte deux champs : un champ clé et un champ contenu. WinTool™ ne permet pas d'avoir plusieurs champs clé pour un même enregistrement.

Sous WinTool™, deux bases au maximum peuvent être ouvertes simultanément. Mais il n'est possible d'effectuer une recherche que dans une base à la fois, celle qui est active, signalée par une marque particulière lorsqu'on déroule les choix du menu WinTool™. Nous nommerons "coche" cette marque particulière ("check mark" dans le jargon Macintosh).

1.3.2.2. WinFile™

WinFile™ est une application sur Macintosh également développée et commercialisée par la société WinSoft [WINF90]. C'est un véritable gestionnaire de fichiers, en ce sens qu'il permet la création de fichiers, leur modification, leur consultation sous différentes présentations et également la recherche dans ces fichiers selon divers critères spécifiés par l'utilisateur.

Contrairement à WinTool™, les fichiers gérés par WinFile™ n'ont pas tous une structure identique. L'utilisateur peut créer une structure propre pour chacun de ses fichiers par une commande appropriée. En particulier, il peut définir comme clés autant de champs qu'il en désire. Il peut aussi visualiser les données d'un fichier selon différentes présentations, par l'intermédiaire de formats qu'il peut créer et modifier à sa guise.

WinFile™ propose également un mode recherche.

1.3.2.3. Word Finder®

Word Finder® de Microlytics™ est aussi un accessoire de bureau sur Macintosh qui permet de consulter des thésaurus anglais-anglais. Ainsi, l'utilisateur a accès à des équivalents d'occurrences anglaises. Les thésaurus sont des fichiers qui ont un format unique et que l'utilisateur ne peut d'ailleurs pas modifier.

Ces fichiers sont accessibles en mode recherche. Depuis l'application courante, l'utilisateur peut copier une occurrence de son texte dans le presse-papier puis appeler Word Finder®. Celui-ci effectue la recherche de l'occurrence dans le thésaurus ouvert et propose une liste de synonymes. L'utilisateur peut alors affiner sa recherche en désignant un des synonymes proposés et en le cherchant à son tour.

Word Finder® maintient à jour une liste des occurrences successivement recherchées depuis le début de la session ; l'utilisateur peut s'en servir pour renouveler une recherche à partir d'une occurrence.

La commande "Replace" permet de remplacer la sélection de l'application courante par le synonyme désigné.

1.3.3. Orientation

Rapidement, nous avons déterminé deux logiciels à exploiter dans une première version d'ODILE. Il s'agit du lemmatiseur extrait de PILAF et de WinTool™. Chacun de ces deux outils représente en effet le surensemble minimal des fonctionnalités désirées.

Ce choix a également été guidé par des considérations de simplicité et de rapidité. Nous voulions en effet concevoir et réaliser notre première version en un temps limité. ATEF était disponible sur IBM sous VM/CMS, l'analyseur de surface développé par P. PALMER était disponible sur VAX 11/780 et GOULD UTX/32 sous UNIX. Or, le lemmatiseur extrait de PILAF, contrairement aux autres analyseurs présentés, était disponible sur micro-ordinateur IBM PC en Turbo Pascal version 5.0 et avait déjà fait l'objet d'un premier portage partiel, sur Macintosh. Turbo Pascal n'étant plus suivi sur Macintosh, nous avons décidé de procéder à un deuxième portage, à partir de la dernière version sur PC, dans un Pascal légèrement différent, THINK Pascal™ .

La tâche était relativement simple également en ce qui concerne WinTool™, qui fournit les fonctionnalités minimales requises par ODILE. Un avantage non négligeable était de pouvoir bénéficier du concours des concepteurs du logiciel, notamment pour définir avec eux l'interface entre WinTool™ et ODILE.

WinFile™ a l'avantage d'autoriser l'existence de plusieurs clés pour un même enregistrement, ce qui, dans le cadre du projet, permettrait de discriminer certains résultats de lemmatisation, avant l'accès à la documentation, lorsque l'analyse d'une occurrence donnée donne un résultat multiple. Cependant, cet avantage n'est pas suffisant pour entreprendre

l'intégration d'un logiciel complexe qui offre bien plus que les fonctionnalités dont nous avons besoin.

Quant à Word Finder®, il ne permet pas la gestion des dictionnaires qu'il utilise. Cela n'est pas acceptable dans O_DI_LE.

2. Objectifs et contraintes

Même si nous avons choisi les logiciels à intégrer dans la première version d'O_DI_LE, il n'en reste pas moins que notre outil doit pouvoir être adapté pour utiliser d'autres lemmatiseurs et d'autres outils dictionnaires.

C'est d'ailleurs pour cette raison que l'outil final a été baptisé O_DI_LE,

Outil d'**I**ntégration **E**xtensible de **D**ictionnaires et de **L**emmatiseurs.

2.1. Objectifs

2.1.1. O_DI_LE

2.1.1.1. Ergonomie de l'outil, structuration du système interactif

Dans [COUT88], J. Coutaz insiste sur la nécessité, lors de la conception d'un système, de prendre en compte les besoins et les limites cognitives de l'utilisateur. Les fonctions du système doivent être de nature à compléter les facultés de l'utilisateur, et leur organisation doit correspondre a la structure mentale de résolution, sans quoi l'interaction risque de mener à une situation d'échec.

Cette ergonomie doit être mise en œuvre à travers ce qu'on appelle l'interface utilisateur. C'est le dispositif matériel et logiciel qui établit la connexion physique entre l'utilisateur et le système, et assure les opérations de traduction entre les formalismes de l'utilisateur et ceux du système.

Un de nos objectifs dans ce domaine est donc d'essayer d'appliquer les principes pratiques ergonomiques élaborés au cours des nombreuses études menées à ce sujet. Il s'agit également de guider l'utilisateur, de lui proposer des fonctions, mais en aucun cas de le

contraindre à telle ou telle séquence d'actions, ni de lui interdire telle ou telle commande, sauf si la sécurité est menacée.

Une autre de nos préoccupations est de concevoir un système interactif qui autorise l'évolution et plus précisément, une évolution rapide et simple à mettre en œuvre. Cela impose de définir une structure particulière du système interactif.

2.1.1.2. "Propreté" de l'intégrateur

ODILE doit pouvoir intégrer plusieurs lemmatiseurs et divers outils d'accès à des bases documentaires. La facilité d'évolution du logiciel pourra être garantie par sa conception modulaire et la définition claire du rôle de chaque module. Une conséquence de ce choix est que les logiciels préexistants concernés ne doivent pas être modifiés. Satisfaire cette contrainte implique la définition d'interfaces générales de deux types :

- une interface entre les différents lemmatiseurs intégrés et ODILE,
- une interface entre les différents outils dictionnaires et ODILE.

2.1.1.3. Généricité de l'outil

L'objectif principal de ce projet est, avec celui de fournir à l'utilisateur un outil linguistique qui lui faisait défaut jusqu'à présent, de concevoir une sorte "d'outil caméléon" qui puisse s'adapter au contexte d'utilisation.

Nous avons vu que les lemmatiseurs traitent plus ou moins bien certains problèmes, qu'ils ont leurs limites (traitement des mots composés, analyse de contexte, etc.) et qu'ils sont globalement tous insatisfaisants. De façon symétrique, les outils dictionnaire, outre leurs fonctions de recherche et de gestion des dictionnaires, ne proposent pas de services supplémentaires équivalents. L'utilisateur doit donc avoir la possibilité de choisir, parmi les lemmatiseurs et les outils dictionnaire intégrés à ODILE, ceux qui sont les plus aptes à résoudre son problème.

Imaginons qu'ODILE intègre n lemmatiseurs Lem_1, Lem_2, ..., Lem_n et p outils d'accès à des bases documentaires Dic_1, Dic_2, ..., Dic_p, ODILE peut engendrer alors n*p outils différents correspondant aux n*p couples possibles de type (Lem_i,Dic_j).

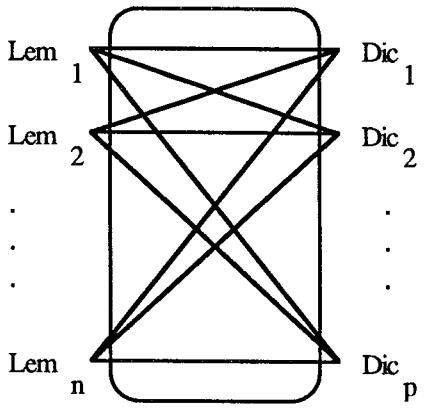


figure 1.4

A tout moment, l'utilisateur pourra choisir le couple (Lem_i,Dic_j) le mieux adapté pour résoudre son problème.

2.1.2. Le lemmatiseur

Dans la première version d'ODILE, nous avons choisi d'intégrer le lemmatiseur extrait de PILAF.

Il apparaît cependant que les possibilités actuelles du lemmatiseur tiré de PILAF sont trop limitées pour espérer fournir un résultat satisfaisant dans un contexte d'utilisation par le grand public, notamment en terme de temps de génération des lemmes et de place occupée par les dictionnaires. Un des buts de notre travail a donc été d'étudier les améliorations qu'on pourrait apporter à ce lemmatiseur, en particulier au niveau de la structure de ses dictionnaires et de la définition des contraintes de lemmatisation.

2.1.2.1. Structure des dictionnaires de PILAF

L'équipe TRILAN du LGI a actuellement trois versions du dictionnaire du français. Le premier, que nous avons utilisé pour les tests, comporte 3000 entrées, et occupe 170 Koctets sur disque. Le second comporte 35000 entrées, peut générer environ 250000 formes, et sa structure a été compactée : il n'occupe plus que 1260 Koctets (au lieu de 2000 Koctets environ). Cependant, il n'est plus modifiable. Le troisième, de 35000 entrées également, a été compacté d'une autre manière et occupe environ 955 Koctets sur disque : il est modifiable, mais pas toujours de manière satisfaisante, particulièrement en cas de débordement du fichier dans lequel on insère une base. Pour plus de détails sur la procédure d'insertion d'un élément dans le dictionnaire PILAF, on pourra consulter [CHAP89b] première partie, paragraphe 5.4. Nous étudions plus en détail ces structures au paragraphe 4.1.3.1. du chapitre II.

En fait, vu la stratégie linguistique utilisée (pas de traitement des mots composés ni des mots inconnus), il faudrait que le dictionnaire comporte au moins 100000 entrées pour une utilisation efficace sur des textes arbitraires. Il nous faut donc réfléchir à des organisations encore plus compactes, qui permettent toutefois des modifications incrémentales. En effet, il importe que les utilisateurs "initiés" puissent compléter le dictionnaire, qui ne sera jamais complet, même s'il est déjà très grand.

2.1.2.2. Les contraintes de lemmatisation

Dans le paragraphe 1.3.1.1., nous avons évoqué le projet MacPILAF de développement d'un lemmatiseur, sur Macintosh, à partir du logiciel PILAF. Le constat fait au terme du projet a mis en évidence la nécessité de procéder à certaines améliorations [CHAP89b]. Il est apparu notamment que la structure des contraintes de lemmatisation utilisée jusque là n'était pas suffisante pour résoudre certains problèmes, en particulier :

- le problème des substantifs ayant un lemme au pluriel

  ex. "gens" est un substantif qui a pour lemme un substantif au pluriel.

  Or, la contrainte de lemmatisation qui s'applique est celle qui, à un substantif, fait correspondre un substantif singulier,

- le problème des substantifs ayant deux formes

  ex. "acteur" -> lemme "acteur"
  "actrice" -> lemme "acteur".

  Il n'y a pas, dans l'état actuel, de contrainte qui permette de faire correspondre, dans certains cas seulement, un lemme substantif masculin à une occurrence substantif féminin.

Il convient donc de raffiner la notion de contraintes de lemmatisation pour se libérer de l'association univoque classe lexicale/contrainte.

2.1.3. L'outil d'accès au dictionnaire

L'objectif concernant l'outil d'accès aux dictionnaires usuels est un objectif de rusticité. Au premier abord, cela peut paraître paradoxal. En effet, les logiciels actuels visent généralement à fournir à l'utilisateur des services perfectionnés et rapides à mettre en œuvre.

Ici, le but est différent. Nous devons intégrer des services divers et complémentaires, mais que l'utilisateur pourra également utiliser indépendamment. L'utilisateur habitué au fonctionnement d'un outil dictionnaire susceptible d'être intégré dans ODILE doit pouvoir

retrouver exactement les mêmes fonctionnalités activées par les mêmes commandes et avec la même présentation lorsque cet outil est intégré dans ODILE.

2.2. Contraintes

2.2.1. Aspects matériels

Pour la réalisation de notre outil, l'utilisation du Macintosh était une contrainte de départ. Son mode opératoire, faisant appel à la souris, à des icônes, à des menus déroulants, à des fenêtres, etc, et sa présentation graphique, permettant des présentations agréables et des instructions d'emploi imagées, en font une machine très attrayante pour l'utilisateur.

L'expérience a prouvé le succès des logiciels développés sur Macintosh. Ce succès a plusieurs raisons. Apple Computer Inc a, d'une part, définit une norme pour l'interface utilisateur de tout logiciel développé sur Macintosh. Cette norme est difficilement contournable si bien que tout logiciel a une forme standard prévisible. En conséquence, l'apprentissage de nouveaux logiciels est relativement rapide. D'autre part, après avoir compris un certain nombre de mécanismes de bases (notion de fichier, mécanisme de sauvegarde,...), point n'est besoin d'être informaticien pour les utiliser.

Le choix du Macintosh semble donc être une bonne idée, mais il induit un certain nombre de contraintes sur le développement d'ODILE. Il est indispensable d'adapter les programmes à intégrer préexistant sur une autre machine aux spécificités du Macintosh, et à sa gestion de la mémoire particulière.

2.2.2. Aspects logiciels

Le langage de programmation qui a été utilisé pour développer les divers modules à intégrer est Pascal, mais un Pascal différent selon les modules : Turbo-Pascal pour PILAF, TML-Pascal pour WinTool™. Il est nécessaire d'avoir un langage commun. Une contrainte supplémentaire est alors de devoir porter PILAF ou WinTool™ ou les deux modules vers ce langage commun. Il paraît évident de choisir Pascal pour réaliser l'intégrateur pour des questions de rapidité et de simplicité. Il reste à déterminer quel Pascal.

Il est à noter ici que la contrainte du langage commun n'en est pas vraiment une. Nous aurions en effet pu mettre les modules à intégrer sous forme de ressources "CODE" sur Macintosh. Le langage dans lequel est exprimé chaque module est alors indifférent. Un autre avantage est que l'utilisation de telles ressources est un moyen de segmenter les gros

programmes : cela est appréciable dans la mesure où les environnements de développement qui permettent la segmentation des accessoires de bureau sont encore peu nombreux.

Par contre, il existe une contrainte : il faut écrire des petites routines en assembleur pour faire le lien entre le programme principal et la ressource "CODE" sollicitée.
Nous avons donc préféré la solution du langage commun, à savoir Pascal.

Quant au système interactif sur lequel sera fondé ODILE, il faut trouver une architecture telle que sa maintenance soit aisée. En particulier, il faut définir une organisation des traitements telle qu'une modification dans la présentation des écrans à l'utilisateur ne remette pas en cause les calculs internes et vice-versa.

Il s'agit également de structurer l'interaction, c'est-à-dire de définir une hiérarchie d'environnements, en prenant soin de trouver un compromis entre les exigences de l'ergonomie et celles de l'intégration [COUT88]. Par exemple, dans le cadre de notre projet, nous serions tentés de "remonter" certaines commandes fréquentes et/ou fondamentales de certains modules intégrés dans ODILE au niveau du menu général ODILE, ainsi que les principes pratiques ergonomiques le préconiseraient. Or, cette organisation des commandes irait à l'encontre du principe selon lequel les logiciels intégrés ne doivent pas être modifiés et leurs commandes ne doivent pas être dupliquées par des commandes ODILE.

2.2.3. Aspects temporels

Compte tenu du temps disponible (onze mois), nous avons été amené à définir une version minimale de l'outil, réalisable dans ce délai, et à étudier séparément les améliorations envisageables dans des versions ultérieures.

2.3. Choix

Concernant le langage de développement, nous avons choisi THINK Pascal™, pour ses nombreux avantages [THIN90]. L'environnement de développement qu'il propose est extrêmement pratique. Il intègre non seulement un compilateur et un éditeur de liens très rapides, mais encore un éditeur, particulièrement approprié à la syntaxe Pascal, qui met en forme les programmes entrés au clavier et contrôle automatiquement leur syntaxe.

THINK Pascal™ propose également des outils de mise au point perfectionnés et un gestionnaire de projet qui maintient les liens entre les différentes unités d'une même application.

THINK Pascal™ est capable d'effectuer des recompilations sélectives, c'est-à-dire la recompilation des seuls éléments du projet qui ont été modifiés.

En ce qui concerne les accessoires de bureau, qui sont des applications particulières sur Macintosh, il permet :

- qu'ils contiennent des déclarations de variables globales,
- qu'ils aient une taille supérieure à 32 K octets,
- qu'ils soient segmentés,

ce qui constitue des atouts non négligeables.

THINK Pascal™ propose également des facilités pour la mise au point des accessoires de bureau, notamment en permettant de simuler leur comportement à travers une application spéciale, le DA Shell.

Concernant le matériel destiné à supporter le logiciel développé, notre choix s'est porté vers le Macintosh Plus qui est la configuration minimale de la gamme Macintosh avec 128K de mémoire morte et un mégaoctet de mémoire vive.

La finalité du logiciel que nous voulons développer est de toucher la population la plus vaste possible. Or, les micro-ordinateurs de la gamme Macintosh sont relativement plus chers que les compatibles PC. Concevoir l'intégrateur pour un matériel plus puissant reviendrait à en réserver son utilisation à une élite, ce qui n'est pas le but visé.

Chapitre II
Conception

Nous avons choisi de concentrer nos efforts sur trois aspects principaux.

L'intégrateur ODILE présente trois faces de communication avec l'extérieur. La première est dédiée aux échanges d'informations entre ODILE et l'utilisateur via une interface appelée interface utilisateur. Les deux autres permettent les échanges avec les logiciels intégrés via des interfaces programmes.

Le premier aspect concerne la conception de l'interface utilisateur du système interactif. Il met en évidence la nécessité d'observer un certain nombre de règles dans cette phase, pour assurer au système final une bonne qualité ergonomique et la possibilité d'une maintenance aisée.

Le second aspect met l'accent sur le problème posé par l'intégration de logiciels préexistants. Plus précisément, cette intégration nous conduit à définir des interfaces programmes générales pour un lemmatiseur quelconque et pour un outil dictionnaire quelconque.

Le troisième aspect concerne l'amélioration de la structure des dictionnaires du lemmatiseur extrait de PILAF. Dans la troisième partie nous étudions donc différentes organisations de fichiers. Cette étude nous permet de dégager celle qui semble la mieux adaptée à nos besoins.

1. Introduction

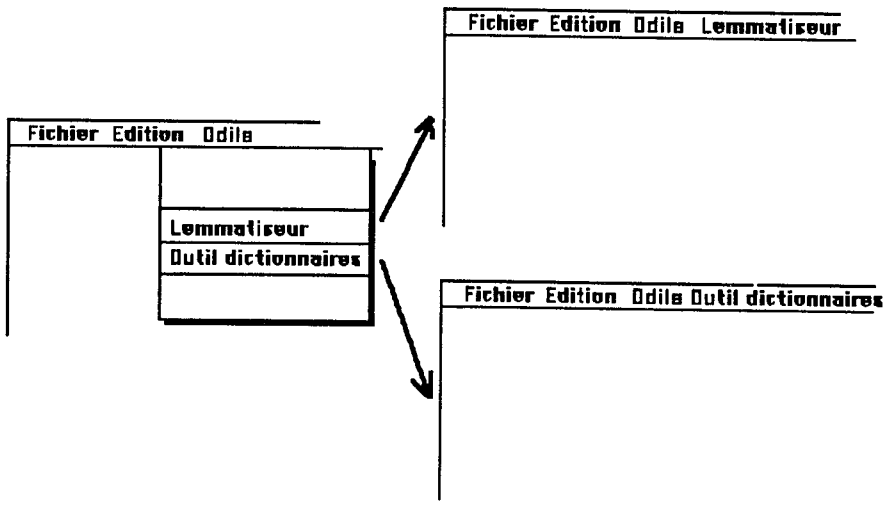
Nous évoquons ici les problèmes soulevés par le choix de l'architecture générale d'ODILE.

1.1. Problèmes rencontrés

Au départ du projet, notre souhait essentiel était d'obtenir un outil générique, et plus précisément, un outil pouvant utiliser pour la lemmatisation et pour l'accès aux dictionnaires, respectivement n'importe quel logiciel de lemmatisation disponible sur Macintosh et n'importe quel outil dictionnaire sur Macintosh.

Nous pensions donc proposer à l'utilisateur un dialogue lui permettant de fixer, au démarrage d'ODILE, le lemmatiseur et l'outil dictionnaire à utiliser.

Quant à la forme générale de l'outil, elle paraissait évidente. L'outil devait pouvoir être activé depuis n'importe quelle application manipulant du texte : l'accessoire de bureau s'imposait. En cours d'exécution, le recours aux fonctions de lemmatisation ou aux fonctions dictionnaire, à travers l'activation d'une commande spécifique du menu ODILE, se concrétiserait par l'appel de l'accessoire de bureau réalisant ces fonctions, et se traduirait graphiquement par la montée du nom de cet accessoire dans la barre de menu.



1.1.1. Problèmes liés à la forme de l'outil

Mais nous nous sommes heurté à un premier problème, d'ordre technique : il n'est pas possible de réaliser l'appel d'un accessoire de bureau depuis un accessoire de bureau en respectant les normes de programmation standard du Macintosh. Cela nécessiterait en effet une

gestion particulière des événements allant à l'encontre des principes de gestion des événements prévus par le système d'exploitation du Macintosh. Il aurait alors fallu avoir recours à un certain nombre "d'astuces de programmation" qui auraient augmenté considérablement le temps de développement et auraient risqué de mettre sérieusement en péril la portabilité et la pérennité du produit final.

Pour contourner ce problème, nous avons ensuite envisagé de concevoir ODILE comme une application sous MultiFinder, le système multitâche sur Macintosh. Ce système permet l'ouverture simultanée de plusieurs applications : par exemple, on aurait pu ouvrir ODILE en même temps qu'un traitement de texte.

Dans ce contexte, le problème d'appel des accessoires de bureau aurait été résolu puisque leur appel depuis une application est tout à fait standard. Cependant, tout n'est pas réglé. Dans notre outil, la communication entre l'application appelante et les accessoires de bureau activés par ODILE est réalisée par l'intermédiaire d'une chaîne de caractères sélectionnée. Or, plusieurs autres applications peuvent être ouvertes et contenir également des sélections. Quelle est la sélection à prendre en compte? Il faudrait encore dans ce cas avoir recours à des astuces pour remédier à ce problème. Pour les raisons évoquées plus haut, nous avons écarté cette idée.

Restait alors à revenir à la solution de l'accessoire de bureau dans une forme légèrement différente : le choix "Lemmatiseur" ou "Outil dictionnaire" dans le menu ODILE aura pour effet l'activation du module correspondant intégré cette fois-ci dans l'accessoire de bureau ODILE. En raison de la contrainte du standard Macintosh selon laquelle un accessoire de bureau ne peut posséder qu'un seul menu, l'utilisation des menus hiérarchiques est impossible. Les commandes prévues par chacun des modules intégrés seront proposées dans une fenêtre de dialogue. De même, le choix d'articles du menu ODILE qui proposent un ensemble de commandes à l'utilisateur provoque l'affichage d'une fenêtre de dialogue. Cela est exposé plus en détail au paragraphe 2.2.1. (Spécifications externes d'ODILE).

1.1.2. Problèmes liés à l'intégration de modules

Nous avons rencontré un deuxième problème, lié à l'interfaçage des modules lemmatiseur et outil dictionnaire avec ODILE.

- Interfaçage avec un lemmatiseur

Sachant que les différents moteurs utilisables sont le fruit de réalisations internes (GETA, LGI, autres laboratoires), ils peuvent être adaptés sans trop de problèmes à nos besoins, c'est-à-dire à une standardisation de leurs entrées et de leurs sorties. De cette manière, l'interface

programme consistant à faire le lien entre ODILE et le lemmatiseur peut être définie une fois pour toute et tout lemmatiseur, moyennant quelques adaptations, pourra être couplé à ODILE.

- Interfaçage avec un outil dictionnaire

S'agissant là d'outils du commerce, il n'est pas possible d'adapter les structures de communication de ces logiciels à nos exigences. Pour utiliser chacun des logiciels d'accès à des dictionnaires, il faut définir, pour chaque logiciel, une interface particulière tenant compte de ses spécificités en matière de recherche et en matière de gestion de fichiers.

En matière de recherche, tous les outils dictionnaire étudiés (chapitre I paragraphe 1.3.2.) recherchent une seule clé à la fois, et c'est le mode de fonctionnement qui semble le plus naturel.

Mais, comme nous le détaillerons plus loin, ODILE et l'outil dictionnaire communiquent aussi par la fonction de gestion de fichiers ("Gérer les clés personnelles..." page 54). Or, d'un outil dictionnaire à l'autre, le format des enregistrements des fichiers gérés peut être différent (WinFile™ page 23 chapitre I). Ces spécificités impliquent la définition d'interfaces différentes. En conséquence, il conviendrait de prévoir une interface sous-découpée en de multiples parties. Chacune des parties serait adaptée à un type différent d'outil dictionnaire.

En outre, l'idée de paramétrer l'outil dictionnaire utilisé dans ODILE ne semble pas être commercialement raisonnable. Cela ferait d'ODILE un logiciel énorme et difficilement maîtrisable. Par contre, créer une version différente d'ODILE pour chaque outil dictionnaire intégré semble être réaliste, d'autant plus qu'on peut souvent transférer un dictionnaire d'un outil à un autre en utilisant les fonctions standard d'import/export.

C'est pourquoi, dans notre prototype, nous abandonnons le paramétrage par l'outil dictionnaire. Nous avons choisi l'outil dictionnaire, en l'occurrence WinTool™, et nous comptons définir une interface programme clairement délimitée. De cette façon, l'adaptation d'un autre outil dictionnaire se résume à une modification éventuelle de l'interface programme. On peut ainsi envisager d'autres versions d'ODILE qui utiliseraient WinFile™, Word Finder®, etc.

Nous aboutissons donc au compromis suivant : dans cette version, ODILE sera un accessoire de bureau mi-paramétrable (lemmatiseur), mi-adaptable (outil dictionnaire).

2. L'interface homme-machine

Dans cette partie, nous définissons le rôle d'ODILE, et plus précisément comment cet intégrateur utilise et complète les fonctionnalités des logiciels intégrés. Puis, nous montrons comment mettre en œuvre ODILE à travers la présentation de son interface utilisateur. Nous étudions enfin une méthode de modélisation du système interactif ODILE.

2.1. Spécifications abstraites

2.1.1. Principe de génération de clés de recherche

L'accès à des dictionnaires de documentation ou de traduction à partir d'éléments de texte exprimés en langage naturel pose un problème : les chaînes de caractères sélectionnées sont rarement directement des entrées dans ces dictionnaires. L'intérêt d'ODILE est de fabriquer et de proposer à l'utilisateur un ensemble de clés potentielles, c'est-à-dire susceptibles d'être des formes standard ou autrement dit des entrées dans ces dictionnaires. L'utilisateur peut choisir parmi celles-ci avant d'accéder aux dictionnaires usuels.

L'élaboration du processus de constitution des clés a été progressive.

Une **première approche** consiste à faire appel aux services d'un lemmatiseur. Lorsque la chaîne ne contient qu'une occurrence, les clés potentielles calculées sont l'ensemble des lemmes déterminés dans l'ordre chronologique de l'analyse. Elles sont souvent satisfaisantes ce qui signifie que parmi celles-ci figure au moins une entrée réelle du dictionnaire usuel.

Exemple :

Si la chaîne sélectionnée est "fermes", l'ensemble des analyses possibles, pour un lemmatiseur simple (ne fournissant pas de catégorie syntaxique), peut être :

ferme	nom féminin	lemme	fermé
ferme	adjectif	lemme	ferme
ferme	verbe	lemme	fermer

l'ensemble de clés proposé est : ferme
fermer

Mais plus fréquemment la chaîne de caractères contient plusieurs occurrences ; c'est entre autres souvent le cas pour la traduction de documentation technique. Le lemmatiseur, qui travaille occurrence par occurrence, fournit alors une liste de clés potentielles fabriquées à partir de combinaisons des lemmes de chacune des occurrences, de la manière suivante :

clé ::= lemme {lemme}

Chaque "lemme" est un élément d'une liste déterminée, pour chaque occurrence, comme plus haut . Les clés potentielles obtenues de cette façon sont souvent insatisfaisantes (clé "patin à roulette" pour la chaîne "patins à roulettes"). Les clés constituées d'une suite de lemmes n'ont généralement aucune pertinence, en particulier dans le contexte de la traduction de documentation technique où ce sont essentiellement des expressions composées qui posent problème au traducteur.

Une **deuxième approche** vise à obtenir des clés potentielles plus réalistes. Elle pallie, d'une certaine manière, le jeu inévitable entre les deux outils indépendants que sont le lemmatiseur et l'outil d'accès aux dictionnaires. En effet, les clés proposées par le lemmatiseur sont des clés potentielles de recherche dans les dictionnaires et ne correspondent pas toujours exactement aux entrées réelles de ces mêmes dictionnaires. La deuxième approche consiste donc à combiner les occurrences d'origine et les lemmes calculés pour chaque occurrence, de la manière suivante:

clés ::= (lemme / forme) {lemme / forme}

"forme" est une occurrence dans la chaîne sélectionnée, et “lemme” est déterminé selon la méthode vue plus haut . On pose d'autre part : lemme < forme.

Exemple :

Si la chaîne sélectionnée est : "patins à roulettes",

les clés issues de la combinaison sont : "patin à roulette"
"patin à roulettes"
"patins à roulette"
"patins à roulettes".

De cette manière, parmi les clés potentielles proposées, nous obtenons l'entrée réelle "patin à roulettes" (un patin a toujours plusieurs roulettes). Il est à noter que si la combinaison fait apparaître des redondances, nous ne proposons à l'utilisateur que les clés potentielles différentes entre elles.

Cette méthode, qui retient non seulement les lemmes, mais aussi les formes d'origine, pour la constitution des entrées de dictionnaire, permet de remédier au problème de l'association univoque classe lexicale/contrainte de lemmatisation évoqué au chapitre I.

Exemple :
Considérons la chaîne d'entrée "gens". C'est un substantif commun au masculin pluriel.
Les contraintes de lemmatisation introduites par le projet MacPILAF nous disent qu'à un substantif correspond un substantif au singulier. Or le lemmatiseur ne trouvera aucune forme dérivée de "gens" correspondant à ces critères, donc pas de lemme. ODILE construira cependant l'entrée "gens" à partir de la forme d'origine, et la recherche réussira.

Cette deuxième approche utilise les résultats de la phase de calcul des lemmes du lemmatiseur, mais est à la charge de l'intégrateur ODILE. Elle ne permet cependant pas d'avoir un résultat satisfaisant dans tous les cas, en particulier en raison de la variété des entrées possibles d'un dictionnaire à l'autre.

Exemple :

Si nous prenons la chaîne suivante :	"République Fédérale Allemande",
l'ensemble des clés proposées est :	"République Fédérale Allemande",
	"République Fédéral Allemand",
	"République Fédérale Allemand",
	"République Fédéral Allemande".

En supposant que nous puissions consulter un dictionnaire des pays contenant la clé "République Fédérale d'Allemagne" et un dictionnaire plus général contenant la clé "RFA", nous n'obtenons l'entrée souhaitée pour aucun des dictionnaires disponibles.

Pour résoudre ce cas, nous avons donc adjoint la possibilité pour l'utilisateur de compléter lui-même la liste des clés potentielles par simple saisie manuelle. Dans l'exemple présenté ci-dessus, il pourra ajouter les clés "République Fédérale d'Allemagne" et "RFA".

Pour le français, la catégorie d'un groupe d'occurrences est en général celle de la première occurrence du groupe. Par exemple, dans l'expression "poêle à frire", la catégorie du groupe est substantif, et non verbe. Cette remarque est importante si l'on envisage, dans une version ultérieure, d'affiner le résultat de la fonction de lemmatisation, c'est-à-dire de ne plus avoir seulement une liste de lemmes, mais également les catégories syntaxiques associées à chacun de ces lemmes. Cette information supplémentaire permettrait à l'utilisateur de discriminer certaines clés en ne demandant la recherche, dans le ou les dictionnaires usuels, que de celles dont la catégorie lui convient.

Dans ces conditions, il conviendra de prévoir une structure, gérée par ODILE, qui devra mettre en correspondance les catégories définies dans le module de lemmatisation et celles que l'on pourra rencontrer dans les entrées des dictionnaires usuels, les catégories dans les deux entités n'étant pas forcément identiques.

Enfin une **troisième approche** pour constituer la liste des clés potentielles est d'utiliser des clés potentielles déjà calculées.

L'idée est donc de pouvoir mémoriser dans des répertoires personnels la ou les clés intéressantes pour une chaîne de caractères donnée. Chaque élément de répertoire est constitué d'une partie identificateur, la chaîne de caractères originelle, et d'un contenu, la liste des clés potentielles mémorisées pour cette chaîne. C'est en quelque sorte un trousseau de clés. Dans le cas de la chaîne "République Fédérale Allemande", l'utilisateur peut par exemple mémoriser l'élément d'identificateur "République Fédérale Allemande" et de contenu les clés "République Fédérale Allemande", "République Fédérale d'Allemagne" et "RFA".

Lorsque l'utilisateur rencontrera une chaîne de caractères déjà traitée et pour laquelle certaines clés potentielles de recherche ont été mémorisées, il pourra obtenir ces clés dans son répertoire sans les recalculer selon la méthode exposée dans la deuxième approche.
De cette manière, en reprenant le même exemple,

pour la chaîne de caractères :	"République Fédérale Allemande",
la liste des clés potentielles sera :	"République Fédérale Allemande".
	"République Fédérale d'Allemagne"
	"RFA".

Les répertoires de trousseaux de clés s'enrichissent au fur et à mesure de l'utilisation d'ODILE en fonction de la variété des dictionnaires consultés. Par exemple, on peut imaginer que l'entrée "avion" d'un tel répertoire s'enrichisse des clés potentielles "avion", "aéronef", "appareil", "planeur", "aéroplane", "hydravion".

2.1.2. Les trois classes de dictionnaires

Nous avons vu apparaître, dans la description du principe de génération des clés, différents types de dictionnaires. Nous précisons dans ce paragraphe la spécificité de chacun et, par souci de clarté, nous lui affectons un nom sous lequel il sera désigné dans toute la suite de l'exposé.

Nous nommons "dictionnaires de lemmatisation" les dictionnaires ou fichiers contenant les bases nécessaires à l'analyse morphologique des occurrences composant les chaînes à traiter. Ce type de dictionnaires est manipulé uniquement par le module Lemmatiseur.

Nous appelons "dictionnaires usuels" les fichiers consultés en vue d'obtenir de la documentation ou de la traduction. Nous utilisons l'expression de dictionnaire "usuel" car les

entrées de ces dictionnaires sont identiques dans de nombreux cas aux entrées des dictionnaires papier couramment utilisés. Ces fichiers sont manipulés par le module Outil Dictionnaire.

Nous désignons par les expressions "lexique personnel" ou "fichier des clés personnelles" les fichiers permettant à l'utilisateur de mémoriser les clés fabriquées pour des chaînes d'entrée données. Ces fichiers, de structure similaire à celle des dictionnaires usuels, sont également accédés et gérés par l'Outil Dictionnaire. Nous expliquons ce choix à la fin de ce chapitre au paragraphe 4.1.2.

2.2. Ergonomie

Cette partie présente les spécifications externes d'ODILE que nous avons définies, puis étudie comment celles-ci répondent aux règles essentielles de l'ergonomie.

2.2.1. Spécifications externes d'ODILE

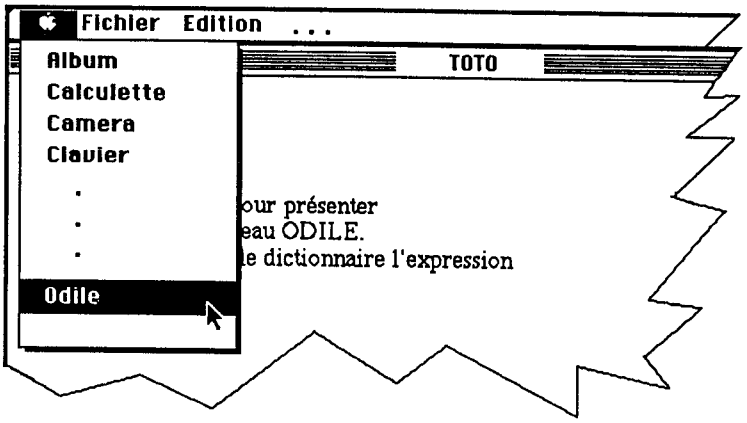
ODILE peut être appelée depuis une application, par exemple un tableur, un système de gestion de bases de données, un traitement de texte, etc. Cette contrainte impose qu'ODILE soit un accessoire de bureau, mis en œuvre dans l'esprit des interfaces Macintosh (fenêtres, icônes, menus déroulants, etc.).

2.2.1.1. Fonctionnalités d'ODILE

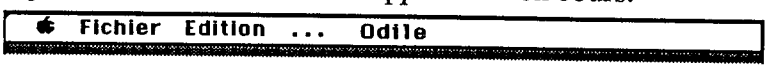
A partir du menu pomme, on sélectionne l'article ODILE, ce qui provoque la montée de son menu déroulant dans la barre de menus. Ce menu comporte des fonctions propres, et permet de retrouver en sous-menus les fonctions du l'outil dictionnaire et du lemmatiseur.

Supposons que l'utilisateur soit sous une application de traitement de texte et que le document "TOTO" soit ouvert.

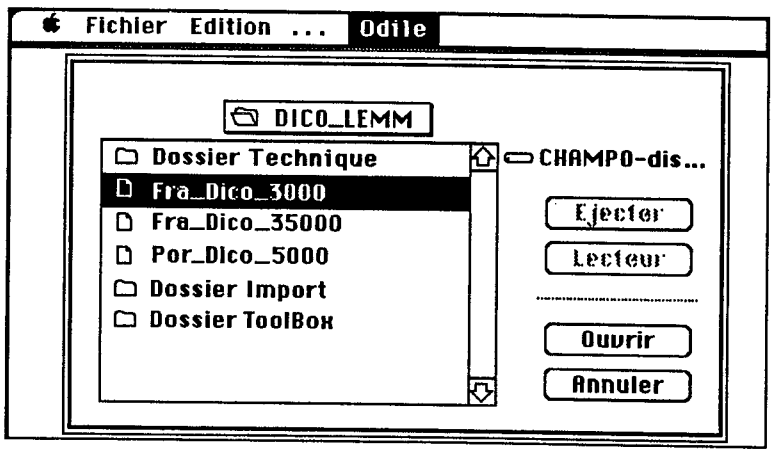
On sélectionne ODILE dans le menu pomme.



Le menu ODILE s'ajoute alors aux menus de l'application en cours.



A l'installation, ODILE propose une fenêtre de dialogue dans laquelle l'utilisateur doit spécifier le dictionnaire qu'il désire utiliser pour la lemmatisation (dictionnaire de bases général, réduit, spécifique à tel ou tel domaine, etc). Il s'agit du dialogue standard d'ouverture de fichier sur Macintosh.



Les lemmatiseurs et plus généralement les analyseurs morphologiques sont souvent des moteurs paramétrés par un dictionnaire et/ou une grammaire (PILAF, ATEF). Les lemmatiseurs fonctionnant à l'aide de dictionnaires utilisent plusieurs fichiers à la fois, et ce qu'on appelle dictionnaire est souvent un ensemble de fichiers physiques. C'est pourquoi le fichier spécifié dans le dialogue d'ouverture de fichier est en fait un fichier permettant d'identifier le contexte d'application du lemmatiseur (dictionnaires et/ou grammaire), c'est-à-dire les fichiers physiques à exploiter. Par exemple, pour PILAF, le fichier "Fra_Dico_3000" sélectionné permet de

déterminer que l'on veut travailler avec l'équivalent du fichier TabMaître (cf la structure du dictionnaire PILAF plus loin, paragraphe 4.1.3.1.) et avec une grammaire du français.

Le choix du contexte d'application du lemmatiseur peut ensuite être modifié (choix "Lemmatiseur..." du menu ODILE page 54). Quant au lemmatiseur lui-même, PILAF est sélectionné par défaut à l'installation de l'accessoire. Ce choix pourra également être modifié par l'activation du choix "Lemmatiseur..." du menu ODILE que l'on exposera plus loin.

Les dictionnaires usuels accédés seront désignés plus loin grâce à un menu fugitif (Dico. Usuel) que nous étudions à la page 53.

Après l'installation, l'utilisateur peut alors mettre en œuvre les commandes du menu ODILE. Ce menu se décompose en un certain nombre de choix possibles correspondant à l'objectif fixé qui est :

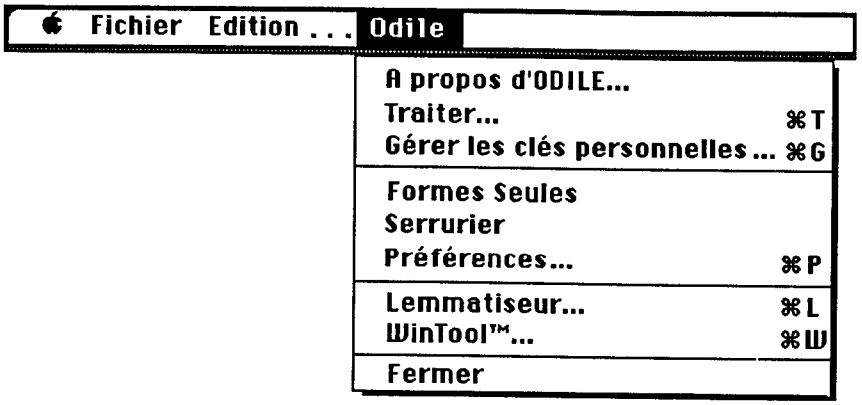
- de privilégier la facilité d'intégration des divers modules participant à ODILE
- de privilégier la forme générique de l'outil .

Dans un premier temps, ce double objectif nous a conduit à séparer, clairement et autant que possible, les fonctions des différents modules, plutôt que de mettre en place des "raccourcis" dans les commandes, de façon à garantir une adaptation rapide et aisée d'ODILE à un autre lemmatiseur que PILAF, et/ou à un autre outil d'accès à des dictionnaires usuels que WinTool™.

Dans un second temps, il s'est avéré que cette conception, satisfaisante du point de vue des principes du génie logiciel, ne l'était pas du point de vue ergonomique, car elle conduisait à contraindre l'utilisateur à effectuer des manipulations beaucoup trop nombreuses.

Nous avons finalement trouvé un compromis et décidé de garder, au niveau de chacun des modules lemmatiseur et outil dictionnaire, ses fonctionnalités spécifiques, sans modification (autant que possible), et d'en dupliquer certaines au niveau d'ODILE (par exemple dans l'article de menu "Traiter...") afin d'améliorer l'ergonomie de l'ensemble.
Au niveau du menu ODILE, on peut donc :

- soit faire communiquer le lemmatiseur et l'outil dictionnaire,
- soit exécuter un certain nombre d'actions préliminaires au lancement d'une fonction d'un des deux modules (options de traitement "Formes Seules" ou "Serrurier", "Préférences...", "Gérer les clés personnelles..."),
- soit, via deux sous-menus, passer sous les modules lemmatiseur et outil dictionnaire.



On peut remarquer que ce menu se décompose en quatre groupes d'articles. Précisons ici qu'un article de menu correspond à une ligne lorsque le menu en question est déroulé. Dans notre cas, le menu ODILE propose neuf articles de menu. Dans la suite de cet exposé, nous appellerons sous-menu chacun des groupes d'articles identifiés.

Nous pouvons ajouter que, conformément aux principes de présentations qui régissent toute interface utilisateur sur Macintosh, les articles de menu qui sont des commandes et qui ouvrent des dialogues lorsqu'ils sont choisis par l'utilisateur sont suivis de trois petits points, les articles de menu qui sont des options de traitement sont signalés par une marque particulière ("check-mark") lorsqu'ils sont activés, c'est-à-dire sélectionnés par l'utilisateur.

Le premier sous-menu contient les articles "A propos d'ODILE...", "Traiter...", "Gérer les clés personnelles...". Ce sont des fonctions propres d'ODILE.

Le second sous-menu contient : "Formes Seules" et "Serrurier". Ce ne sont pas des commandes, mais des options de traitement, que l'utilisateur peut activer ou désactiver par sélection à la souris, et qui sont utilisées pour le calcul des clés de recherche par la commande "Traiter...". Il contient aussi le choix "Préférences...", qui permet à l'utilisateur d'adapter l'interface à ses besoins (voir plus loin).

Le troisième sous-menu propose deux commandes permettant à l'utilisateur d'utiliser les deux logiciels intégrés indépendamment d'ODILE.

La commande "Lemmatiseur..." fait apparaître le menu du lemmatiseur, où l'utilisateur pourra trouver les diverses fonctionnalités du module:

- détermination du ou des lemmes correspondant à la forme sélectionnée,
- affichage des lemmes à l'écran,
- choix du dictionnaire des bases pour la lemmatisation,
- modification des données déterminant la grammaire du lemmatiseur,

- modification du contenu des dictionnaires du lemmatiseur.

Dans une version ultérieure, il serait judicieux de ne rendre accessibles les deux dernières commandes qu'à des utilisateurs autorisés, puisque ces fonctions permettent d'intervenir sur les données conditionnant le bon fonctionnement du lemmatiseur. Pour cela, on définirait des niveaux d'utilisation.

La commande "WinTool™..." fait apparaître une fenêtre spécifique de l'outil d'accès aux dictionnaires usuels, qui permet entre autres :

- d'ouvrir des dictionnaires usuels (2 au maximum actuellement pour WinTool™),
- de sélectionner parmi les dictionnaires ouverts celui sur lequel on veut travailler,
- d'obtenir des informations sur le dictionnaire courant,
- de rechercher un élément du dictionnaire courant ayant une clé donnée,
- d'ajouter, de modifier, ou de supprimer un élément dans le dictionnaire actif.

Il devra contenir en outre une commande permettant la création de fichiers WinTool™ (cette fonction est disponible actuellement dans une application indépendante livrée avec WinTool™).

Le quatrième sous-menu contient un seul choix: Fermer.
Cette fonction inactive ODILE, en fermant les dictionnaires ouverts, les fenêtres ouvertes, et en supprimant le menu ODILE de la barre des menus.

2.2.1.2. Détail du menu ODILE

La majorité des articles du menu ODILE ouvre des fenêtres de dialogue. Ces fenêtres proposent elles-même des commandes.

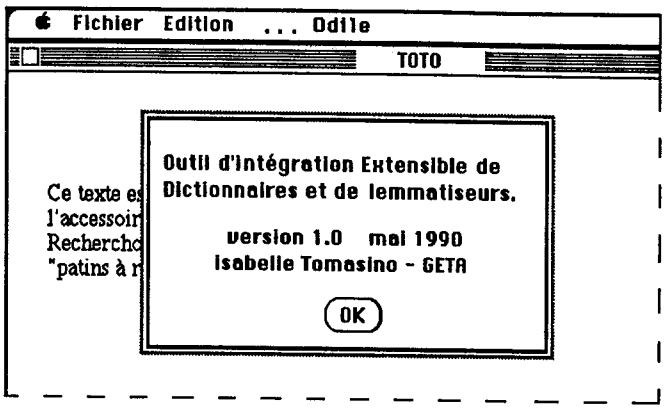
Nous avons présenté l'interface utilisateur d'ODILE de cette façon plutôt que sous forme de menus hiérarchiques car l'interface standard Macintosh n'autorise pas la présence de plusieurs menus dans un accessoire de bureau.

1. Premier sous-menu

("A propos d'ODILE...", "Traiter...", "Gérer les clés personnelles...")

1. "A propos d'ODILE..."

C'est le premier article du menu ODILE. Il donne la carte de visite d'ODILE.

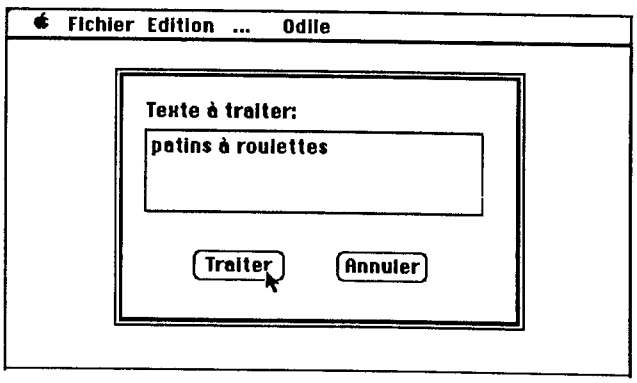


2. *"Traiter..."*

C'est la commande fondamentale d'ODILE. Elle propose une fenêtre de dialogue dans laquelle s'affiche la liste des clés potentielles de recherche calculées à partir d'un groupe d'occurrences sélectionné sous l'application appelante. L'utilisateur peut modifier cette liste. Il peut aussi chercher une clé sélectionnée dans cette liste dans le dictionnaire usuel actif.

Nous aurions pu définir un mode de fonctionnement plus ergonomique où la liste des clés fabriquées serait affichée à la demande en fonction d'un paramètre de l'article de menu "Préférences..." et la recherche de toutes les clés fabriquées dans le dictionnaire usuel serait automatique. Cette solution n'a pas été retenue pour la première version, essentiellement pour des contraintes de temps. Il semble qu'elle soit cependant préférable. Elle fera l'objet d'une prochaine étude en vue d'obtenir une nouvelle version d'ODILE.

Si aucune chaîne de caractères n'a été sélectionnée sous l'application appelante, ODILE ouvre un dialogue dans lequel l'utilisateur peut saisir la chaîne à analyser.



La liste des clés de recherche, déterminée à partir d'une sélection donnée, est obtenue par un calcul effectué selon le principe de génération des clés présenté au paragraphe 2.2.1. de ce

chapitre. Ce calcul dépend des options de traitement actives (sélectionnées par une coche dans le second sous-menu d' ODILE) :

- si l'option "Formes Seules" est inactive, il y a lemmatisation des formes de la chaîne sélectionnée, et les clés potentielles de recherche sont fabriquées par combinaison des lemmes déterminés avec les formes originelles. C'est le cas standard.

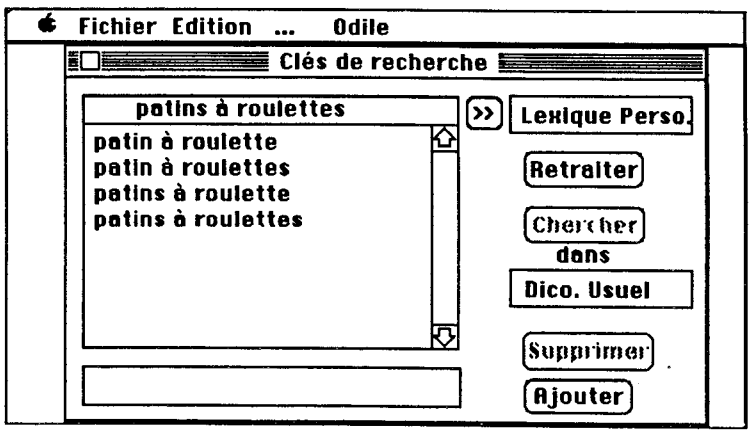
- si l'option "Formes seules" est active, il n'y a pas de lemmatisation. Il y a fabrication d'une seule clé potentielle, la suite des formes de la chaîne d'origine, normalisée (un seul blanc entre deux formes).

- si l'option "Serrurier" est active, il y a recherche dans un fichier de clés personnelles .

Le calcul résultant du choix de l'option "Serrurier" consiste à rechercher la chaîne à analyser comme entrée du lexique personnel actif à cet instant. Si la chaîne est trouvée, ODILE ajoute à la liste des clés de recherche les clés mémorisées pour cette entrée. Pour que ce traitement puisse se dérouler, il faut bien entendu qu'au moins un lexique personnel ait préalablement été installé et soit actif. Dans le cas contraire, le calcul ne peut avoir lieu et un message d'alerte indique à l'utilisateur que l'option "Serrurier" est active mais qu'aucun fichier utilisateur n'est actif.

Comme nous le verrons plus loin, on peut en outre ajouter des clés par saisie manuelle.

Les opérations de calcul décrites s'effectuent dans cet ordre, si et seulement si les options appropriées ont été choisies. Si cela s'avère utile (plus de souplesse pour l'utilisateur), on pourra, dans une version ultérieure, mettre en place un mécanisme tel que les phases de calcul s'enchaînent tant qu'aucune action de l'utilisateur n'intervient. ODILE utilisera le temps CPU inutilisé pour calculer un maximum de clés, mais certaines actions de l'utilisateur (clic-souris dans certains boutons, frappe de certaines touches au clavier) seront prioritaires sur la deuxième phase de calcul, et l'arrêteront.

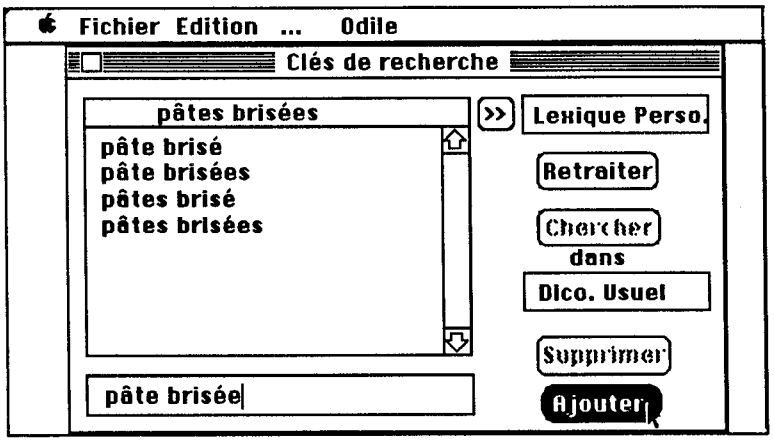


La liste des clés de recherche est constituée des groupes de clés issus des différentes phases de calcul et affichés les uns à la suite des autres dans une fenêtre avec ascenseur.

1. Bouton "Ajouter"

Comme nous l'avons dit plus haut, des clés peuvent être spécifiées manuellement. L'utilisateur saisit la clé qu'il désire ajouter dans la zone de saisie appropriée, puis clique sur le bouton "Ajouter". La clé saisie est automatiquement ajoutée à la fin de la liste.

Dans l'exemple ci-dessous, aucune des clés potentielles fournies dans la liste statique n'est une entrée de dictionnaire. L'utilisateur saisit manuellement dans la zone de saisie prévue à cet effet la clé "pâte brisée" qui lui semble bonne et l'ajoute à la suite des clés de la liste statique.

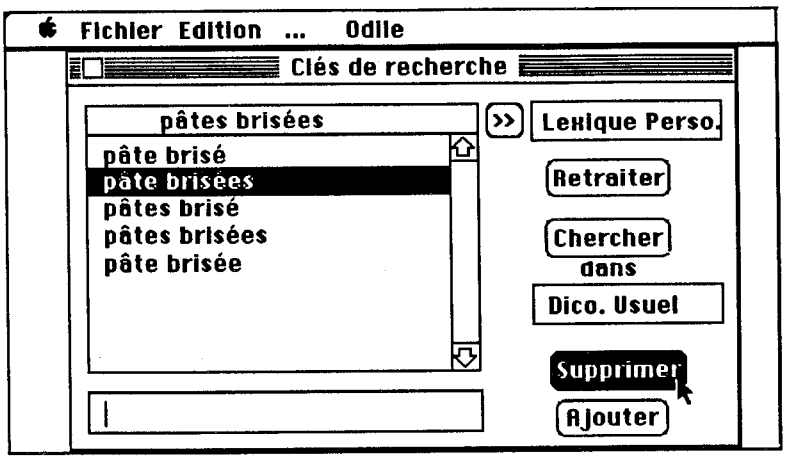


L'utilisateur peut ajouter ainsi autant de clés qu'il le désire.

On pourrait imaginer une autre procédure d'ajout de clés. Cette procédure consisterait dans un premier temps à sélectionner une clé de la liste statique. Cette clé apparaîtrait alors dans la zone de saisie en bas de la fenêtre. Dans un deuxième temps, l'utilisateur n'aurait plus qu'à modifier la clé éditée avant de l'ajouter dans la liste statique. Cette façon de faire faciliterait le travail de l'utilisateur qui désire ajouter des clés très proches de celles dont il dispose déjà dans la liste statique. Elle pourra facilement être implémentée dans la prochaine version d'ODILE.

2. Bouton "Supprimer"

L'utilisateur peut supprimer, dans la liste, des clés qui ne l'intéressent pas, soit pour la recherche dans le dictionnaire usuel, soit pour la mémorisation dans un fichier utilisateur.



Il sélectionne la clé à supprimer dans la liste par simple clic avec la souris. Remarquons ici que les boutons "Supprimer" et "Chercher" ne sont disponibles que lorsqu'une clé est sélectionnée dans la liste. Puis l'utilisateur clique sur le bouton "Supprimer". La clé sélectionnée est supprimée de la liste.

3. Bouton "Retraiter"

Il est possible que des clés que l'utilisateur s'attend à obtenir n'apparaissent pas dans la liste en raison d'un choix particulier d'options de traitement. Dans ce cas, l'utilisateur peut remédier au problème en sélectionnant la ou les options qu'il souhaite activer, dans le menu ODILE, puis en recommençant le calcul des clés pour la même chaîne d'entrée en cliquant sur le bouton "Retraiter".

4. Bouton "Chercher"

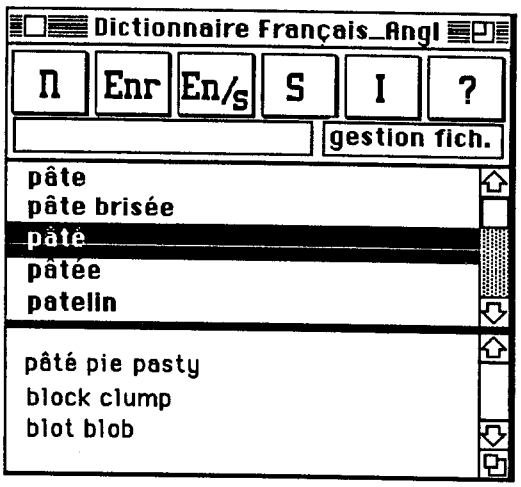
"Chercher" permet à l'utilisateur, en cliquant sur le bouton correspondant, d'activer le processus de recherche de WinTool™. La sélection courante dans la liste de clés affichées est recherchée dans le dictionnaire usuel actif préalablement installé.

En cas de réussite de la recherche, une fenêtre associée au dictionnaire usuel actif est ouverte. Le nom du dictionnaire usuel est le titre de la fenêtre. L'élément de dictionnaire trouvé apparaît dans cette fenêtre : la clé de l'élément apparaît en inverse vidéo dans le cadre supérieur, appelé zone des clés, et son contenu est visible dans le cadre inférieur. C'est la fenêtre résultat de WinTool™.

En cas d'échec, une fenêtre de même nature s'ouvre, mais son contenu est différent. En effet, dans ce cas, c'est l'élément du dictionnaire immédiatement supérieur à la clé recherchée qui est mis en évidence.

Exemple :

Supposons que l'utilisateur souhaite rechercher dans le dictionnaire usuel de traduction "Français_Anglais" la clé potentielle "pâte brisées". Cette clé n'est pas une entrée réelle du dictionnaire. L'utilisateur obtient donc la fenêtre résultat de WinTool™ suivante :



L'utilisateur peut alors consulter la liste des entrées du dictionnaire usuel courant, et peut trouver à proximité la clé qui l'intéresse "pâte brisée" et qu'ODILE n'a pas su fabriquer. On constate donc que l'outil dictionnaire WinTool™ remédie dans certains cas aux limites d'ODILE.

L'utilisateur qui souhaite quitter l'environnement de l'outil dictionnaire pour revenir dans celui d'ODILE peut fermer les fenêtres associées aux dictionnaires usuels ouverts en cliquant dans leur case de fermeture, ou bien directement cliquer dans la fenêtre donnant la liste des clés potentielles d'ODILE.

Actuellement, une seule clé peut être sélectionnée à la fois et recherchée dans le dictionnaire usuel, cela afin d'utiliser la fenêtre résultat de WinTool™ qui ne fournit de résultat de recherche que pour un élément à la fois.

Nous avons envisagé, à la page 48, la possibilité, dans une version ultérieure, de rechercher automatiquement toutes les clés fabriquées les unes après les autres. Dans ce cas, les résultats pourront être affichés dans une fenêtre spécifique d'ODILE qui regroupera tous les résultats issus des recherches effectuées par l'outil dictionnaire.

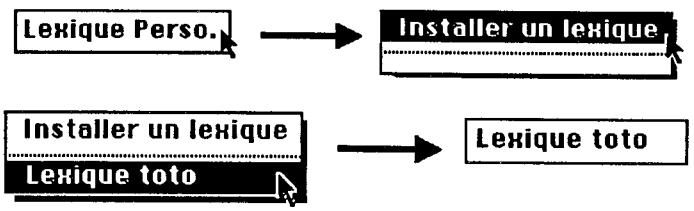
On pourra prévoir, dans ce contexte, une commande permettant d'ordonner différemment les clés potentielles de recherche. Si les clés qui ont apparemment le plus de chance d'être trouvées se situent en tête de la liste et si l'utilisateur sélectionne toutes les clés de la liste pour la recherche, il obtiendra très rapidement dans la fenêtre un résultat. Il pourra stopper la recherche avant d'avoir examiné toute la liste et ainsi gagner du temps.

5. Menus "Lexique perso." et "Dico. Usuel"

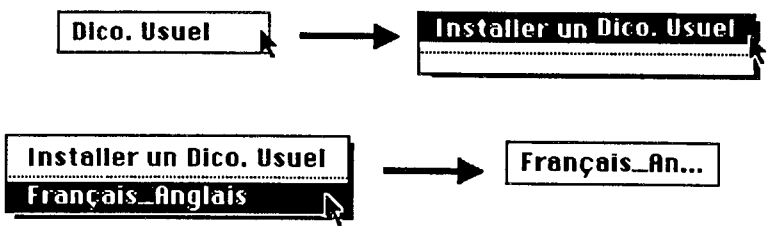
Il nous reste à présenter les fonctionnalités des menus fugitifs (appelés "pop-up" menus dans le jargon du Macintosh) "Lexique perso." et "Dico. Usuel" et du bouton identifié par une double flèche située devant.

Lorsqu'on clique dans le rectangle correspondant au menu fugitif, un nouveau rectangle se superpose et présente un certain nombre de commandes et options que l'utilisateur peut sélectionner. Quand on relâche le bouton de la souris, le rectangle initial se redessine, et contient alors l'item sélectionné.

Dans le menu "Lexique Perso.", la commande "Installer un lexique" permet d'installer un ou plusieurs fichiers de clés personnelles. L'utilisateur peut choisir celui avec lequel il désire travailler en le sélectionnant dans la liste des fichiers installés qui apparaît dans le menu fugitif. Le nom du fichier de clés personnelles courant s'affiche dans le rectangle initial.



Le menu "Dico. Usuel" fonctionne de façon similaire puisqu'il permet d'installer les dictionnaires usuels de format WinTool™ dans lesquels pourra s'effectuer la recherche des clés.



Quant au bouton comportant une double flèche, il sert à alimenter le fichier des clés personnelles actif. Lorsqu'on clique sur ce bouton, l'ensemble des clés contenues dans la liste à

cet instant est mémorisé dans le fichier. La clé d'accès à cet ensemble de clés de recherche, dans le fichier, est la chaîne d'origine affichée en titre de la liste.

3. "Gérer les clés personnelles..."

La mémorisation des clés personnelles suppose l'existence de fonctions capables de gérer des ensembles de clés, quelle que soit leur origine .

Ainsi, il faut pouvoir :

- **ouvrir** un lexique personnel,
- **créer** un nouveau lexique personnel,
- **modifier** le contenu d'un élément du lexique personnel courant,
- **enregistrer sous** un nom de clé donné un élément, dont on a spécifié le contenu, dans le lexique personnel courant (équivalent à créer un élément dans le lexique personnel courant),
- **enregistrer** un élément dans le lexique personnel courant,
- **supprimer** un élément dans le lexique personnel courant,
- **afficher** le contenu d'un lexique personnel.

2. Second sous-menu

("Formes seules", "Serrurier", "Préférences...")

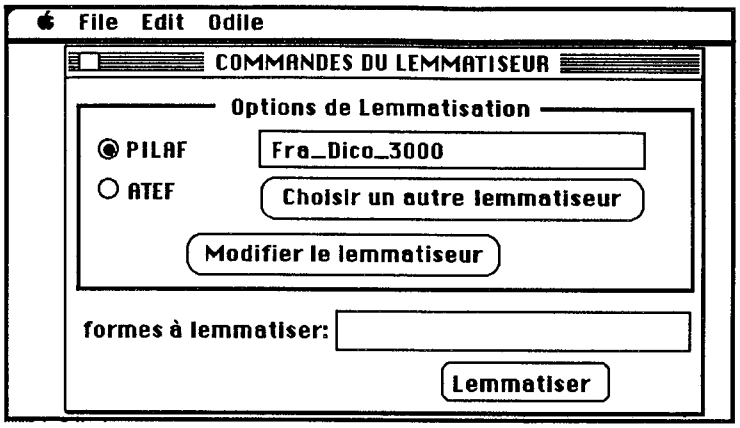
Ce menu propose les deux options de calcul des clés de recherche utilisées par la commande "Traiter..." d'ODILE. Elles ont été présentées plus haut. Nous n'y reviendrons donc pas davantage.

Il contient également l'article "Préférences...". C'est une commande qui ouvre un dialogue permettant à l'utilisateur de personnaliser les équivalents clavier, de rendre visible ou invisible le résultat de la lemmatisation, ou encore de définir le nombre d'occurrences maximum qu'il souhaite pouvoir analyser.

3. Troisième sous-menu ("Lemmatiseur...", "WinTool™...")

1. "Lemmatiseur..."

Choisir l'article "Lemmatiseur..." provoque l'ouverture d'une fenêtre de dialogue proposant les fonctionnalités du module de lemmatisation.



Cette fenêtre s'affiche quel que soit le lemmatiseur utilisé. Elle contient des boutons radio permettant de faire un choix exclusif du moteur de lemmatisation à utiliser (PILAF, ATEF, ...), des boutons de commande, et une zone de saisie pour spécifier la chaîne de caractères à lemmatiser.

Cependant, d'un lemmatiseur à l'autre, les commandes présentées ne sont pas toutes disponibles. Par exemple, la modification des dictionnaires d'ATEF n'est pas prévue de manière incrémentale, et la modification de la grammaire n'est pas prévue de manière interactive : pour ce lemmatiseur, la commande "Modifier le lemmatiseur", qui permet de modifier les dictionnaires et/ou la grammaire du dictionnaire courant, ne sera pas disponible. Une commande est indisponible lorsque du point de vue graphique le bouton correspondant apparaît grisé.

1. Bouton "Choisir un autre lemmatiseur"

Cliquer sur ce bouton affiche à l'écran le dialogue classique d'ouverture de fichier que l'on a déjà vu lors de l'installation d'ODILE (cf page 44). Il permet de spécifier un nouveau contexte d'application du lemmatiseur courant(dictionnaire et/ou grammaire). Il est à noter que le changement de lemmatiseur (en sélectionnant par clic souris le bouton radio correspondant de la fenêtre) provoque également l'affichage du dialogue standard d'ouverture de fichier. Il est destiné à spécifier le contexte d'application du nouveau lemmatiseur.

2. Bouton "Modifier le lemmatiseur"

Par l'intermédiaire de cette commande, et lorsqu'elle est disponible, l'utilisateur autorisé peut, suivant la nature du lemmatiseur, intervenir sur la grammaire et le dictionnaires utilisés dans le contexte d'application courant du lemmatiseur.

Par exemple, pour PILAF, il pourra intervenir sur un certain nombre de types de données de la grammaire: règles, modèles, variables, masques de variables, etc. Les données

de chacun de ces types pourront être créées, modifiées, supprimées. De même, l'utilisateur pourra modifier les dictionnaires. Ces opérations auront lieu à travers l'enchaînement d'écrans spécifiques.

2. *"WinTool™..."*

L'article de menu "WinTool™..." renvoie au menu de l'outil d'accès au dictionnaire, en l'occurrence WinTool™. Nous présentons donc dans ce qui suit la version 1.1 de WinTool™ dont on pourra retrouver toutes les fonctionnalités dans ODILE.

Il est à noter que, dans la documentation de WinTool™, la notion de base est utilisée là où nous employons le terme de dictionnaire. Les deux termes seront employés indifféremment dans les dix points qui suivent.

1. A propos de WinTool...

Ce choix permet de visualiser la carte de visite du produit.

2. Ouvrir une base...

C'est par la commande "Ouvrir une base..." que l'utilisateur peut ouvrir le ou les dictionnaires usuels. Le dialogue standard d'ouverture de fichiers sur Macintosh permet de sélectionner le dictionnaire à ouvrir. La version 1.1 de WinTool™ permet d'ouvrir deux dictionnaires simultanément.

3. Nom 1ère base et Nom 2ème base

Dans le cas d'une utilisation sous ODILE, ces noms indiquent les dictionnaires usuels ouverts.

4. Lire les infos...

Cette commande permet de connaître :
- le nombre d'éléments contenus dans le dictionnaire actif,
- le nombre d'éléments détruits mais toujours présents physiquement dans le dictionnaire,
- le nombre d'octets que l'on pourrait libérer en compactant le dictionnaire.

5. Rechercher...

Cette commande permet de rechercher un élément à partir de la clé qui a été spécifiée dans une boîte de dialogue. Elle n'a en principe pas grand intérêt dans ODILE, puisque, en général, la clé de recherche aura préalablement été calculée par un processus de lemmatisation. Elle reste néammoins disponible.

6. Rechercher la sélection...

Comme son nom l'indique, cette commande permet de faire une recherche à partir d'une sélection quelconque, effectuée avec la souris. La sélection peut, entre autres, être une sélection parmi les clés de recherche potentielles fournies par la fonction de lemmatisation. En raison de contraintes liées à WinTool™, cette commande ne peut rechercher qu'une clé à la fois, et dans un seul dictionnaire (identifié par une “check-mark”).

7. Ajouter une fiche...

Ce choix permet de créer des éléments dans des dictionnaires. Ainsi, l'utilisateur peut construire ses propres dictionnaires. Il peut par exemple créer un répertoire d'éléments particuliers, absents des dictionnaires usuels, ou encore un dictionnaire usuel utilisateur plus général.

"Ajouter une fiche..." permet la création d'un nouvel élément en remplissant le contenu d'un élément vide. Le dialogue de sauvegarde proposé avec la commande "Enregistrer sous..." apparaît dès que l'on quitte l'élément. La clé que l'on fournit permet de classer cet élément par ordre alphabétique dans le dictionnaire choisi. Le nouvel élément devient l'élément courant.

8. Enregistrer...

Cette commande permet d'enregistrer le nouveau contenu d'un élément dont la clé est déjà connue par WinTool™. C'est donc l'équivalent d'une modification.

9. Enregistrer sous...

Propose un dialogue pour définir la clé du nouvel élément que l'on vient d'introduire. Cliquer sur "OK" dans le dialogue provoque l'enregistrement de l'élément sous le nom indiqué.

10. Supprimer la fiche courante

Cette commande permet de supprimer un élément d'une base. L'élément à supprimer aura dû au préalable être sélectionné .

5. Quatrième sous-menu

Fermer

Cette commande inactive ODILE, en le supprimant de la barre des menus et en fermant tous les dictionnaires ouverts (dictionnaires de lemmatisation, dictionnaires usuels ou lexiques personnels) ainsi que tous les fichiers de travail .

2.2.2. Principes pratiques ergonomiques

Ces principes nous sont suggérés dans [COUT88]. Joëlle Coutaz y classe les règles à observer pour la réalisation d'interfaces utilisateurs satisfaisantes en cinq grands thèmes: la cohérence, la concision, les retours d'information, la structuration des activités, et la flexibilité.

Les règles évoquées dans chacun de ces grands thèmes ne sont pas toutes utiles dans ODILE. Nous présentons ici celles qui ont été mises en application.

*** Cohérence**

- aspects lexicaux

Le choix de la nomenclature des commandes est important ; les termes choisis doivent être des termes précis, didactiques et pertinents.

Par exemple, la commande "Traiter..." d'ODILE peut sembler floue. Or, elle est bien adaptée dans la mesure où elle indique à l'utilisateur par sa sémantique générale qu'il s'agit de la commande centrale d'ODILE et lui suggère que toute résolution devra passer par elle.

La nomenclature peut utiliser une métaphore.

L'option "Serrurier" du menu ODILE suggère la fabrication de clés. C'est effectivement ce que réalise cette option puisqu'elle effectue la recherche d'ensembles de clés dans les répertoires de clés que sont les lexiques personnels. Ces métaphores, comme l'explique J. Coutaz, ne réduisent pas la distance articulatoire, c'est-à-dire le temps qu'il faut à l'utilisateur

pour mettre en correspondance l'expression d'entrée (la commande qu'il souhaite exécuter) et la forme de cette expression, mais facilitent certainement l'apprentissage.

Il faut également veiller à l'élimination des incohérences lexicales.
Comme le suggère J. Coutaz, «il faut veiller à ce qu'une fonction sémantique ou un concept soient toujours désignés par le même nom et cela quel que soit le contexte.»

Nous avons mis ce principe en pratique. Dans le menu ODILE, nous proposons le choix "Traiter..." qui calcule les clés de recherche à partir d'une forme préalablement sélectionnée et ouvre un dialogue où sont affichées les clés calculées. A cette étape, le dialogue propose entre autres de recalculer les clés avec d'autres options. Cette commande est matérialisée par le bouton "Retraiter". Nommer ce bouton "Recalculer" ou "Recalculer les clés" aurait été correct, mais aurait introduit une incohérence lexicale puisqu'en fait son rôle est le même que celui du choix "Traiter..." du menu. Il fallait donc garder un nom identique ou du moins très proche sous peine de perturber l'utilisateur.

- aspects pragmatiques

Selon J. Coutaz, la cohérence spatiale aide l'utilisateur à acquérir une connaissance motrice qui permet d'anticiper les actions physiques... Elle aide à franchir la distance articulatoire.»

C'est pourquoi, dans ODILE, les commandes sont proposées dans un menu qui s'affiche toujours au même endroit et dont les commandes apparaissent dans l'ordre logique défini par la tâche que résoud ODILE, à savoir accéder à un dictionnaire usuel à partir d'une forme sélectionnée dans l'application appelante.

Nous trouvons la commande "A propos d'ODILE..." qui donne quelques renseignements d'ordre général sur l'accessoire de bureau, puis suit immédiatement la commande fondamentale "Traiter..." et enfin la commande "Gérer les clés utilisateur...". Ces trois premières commandes sont en tête car il s'agit des commandes spécifiques de l'intégrateur ODILE. Viennent ensuite des commandes annexes, options de calcul des clés, commandes de choix des paramètres d'utilisation et commandes permettant d'activer les logiciels lemmatiseur et outil dictionnaire indépendamment. La commande "Fermer" figure à la fin du menu, puisque c'est sa position traditionnelle dans les accessoires de bureau sur le Macintosh.

Il est à noter, qu'indépendamment d'ODILE, les principes de présentation qui régissent toute interface utilisateur sur Macintosh et qui tendent à définir un style d'interaction homogène participent à cet objectif de cohérence.

* Concision

- abréviations

Elles s'adressent à des utilisateurs expérimentés. Les équivalents clavier proposés sur Macintosh sont destinés à remplacer la désignation d'éléments de menus avec la souris par la frappe d'un caractère équivalent au clavier. Ce genre de facilité est utile dans le sens ou il évite dans certains cas le changement de dispositif de contrôle (passage du clavier à la souris). Il évite aussi de perdre du temps en déroulant des menus.

Ces équivalents clavier sont exploités dans ODILE même si l'application hôte risque de posséder un équivalent clavier identique à celui de l'accessoire de bureau : la frappe du caractère en question au clavier peut alors avoir des conséquences inattendues. Dans ce cas, l'utilisateur pourra recourir au choix "Préférences..." pour modifier l'équivalent clavier d'ODILE qui pose problème.

- valeurs par défaut

Les valeurs par défaut sont des valeurs proposées implicitement ou explicitement par le logiciel. On peut les classer en deux catégories: les valeurs par défaut dynamiques et les valeurs par défaut semi-statiques. Elles permettent de réduire les erreurs typographiques et les erreurs sémantiques.

Les valeurs par défaut dynamiques sont mémorisées et constamment réévaluées par le système. Dans le cas d'une sauvegarde de fichier, le système a mémorisé le nom de fichier en cours d'édition et le propose par défaut comme fichier de sauvegarde.
Les valeurs par défaut semi-statiques ne peuvent être modifiées que par demande explicite de l'utilisateur. Les options de présentation de l'interface utilisateur et les options de l'application entrent dans cette catégories.

ODILE propose des valeurs par défaut semi-statiques. Les équivalents clavier des articles de menu, le nombre d'occurrences traitables, et l'invisibilité des résultats de lemmatisation sont des options de présentation modifiables par l'article de menu "Préférences...". Le lemmatiseur utilisé est une option de l'application modifiable par l'article de menu "Lemmatiseur...".

* Retours d'information

- pour rassurer

Certaines fonctions du système requièrent un temps d'exécution relativement long. L'utilisateur a alors besoin d'être informé sur l'état interne de système, sans quoi les suppositions qu'il fait sur cet état peuvent l'amener à effectuer des opérations injustifiées, voire dangereuses.

Dans ODILE, pendant toute la phase de calcul des clés de recherche, le message " patience! Calcul des clés pour la forme "..." " est affiché à l'écran, et la flèche indiquant la position de la souris est remplacé par la montre, illustrant ainsi le fait que cette opération demande un certain temps.

- pour réduire la charge cognitive

L'interface peut jouer un rôle très important pour libérer l'utilisateur de nombreuses informations qui encombrent son esprit et ralentissent sa capacité à élaborer un plan d'action. L'ordinateur a mémorisé un grand nombre de données et l'interface permet de les transmettre à l'utilisateur. Notamment, elle peut:

. rappeler le contexte de travail, les fichiers ouverts.

Cela est mis en œuvre dans ODILE où le nom du dictionnaire de lemmatisation ouvert est donné dans le dialogue du lemmatiseur, où le nom du dictionnaire usuel et le nom du lexique personnel ouverts sont donnés dans le dialogue issu de la commande "Traiter...".

. présenter les commandes et options de traitement de manière à ce que l'utilisateur sache à tout moment distinguer celles qui sont disponibles de celles qui ne le sont pas.

Dans le dialogue issu de la commande "Traiter..." d'ODILE, lorsqu'aucune clé n'est sélectionnée parmi celles qui sont affichées dans la liste statique, les commandes "Chercher" et "Supprimer" sont désactivées. De même, la commande de mémorisation des clés fabriquées n'est pas disponible tant qu'aucun lexique personnel n'est ouvert. Pour signifier qu'elles ne sont pas disponibles, ces commandes apparaissent en grisé.

. pour détecter des erreurs et proposer des remèdes.

En cas d'erreur le système interactif doit fournir à l'utilisateur un retour d'informations immédiat. Le message doit être suffisamment informatif pour permettre à l'utilisateur d'identifier la cause d'erreur et de trouver le remède.

Ainsi dans ODILE, lorsqu'aucune clé n'a pu être construite à partir de la chaîne de caractères sélectionnée dans le texte de l'application appelante, le message suivant prévient l'utilisateur : "Aucune clé n'a pu être construite à partir de la chaîne de caractères <Chaîne>."

L'utilisateur sait alors qu'il peut relancer le calcul des clés avec d'autres options, ou bien saisir manuellement des clés de son choix, ou bien abandonner le calcul des clés pour cette chaîne.

* Structuration des activités

- structurer le fond

Tout système interactif doit être structuré en une hiérarchie d'environnements, c'est-à-dire en contextes dans lesquels certaines commandes seulement sont possibles. Il s'agit donc de structurer l'espace des commandes en trouvant un compromis : il faut privilégier la rapidité d'expression des commandes les plus fréquentes sans toutefois empêcher l'utilisateur d'exprimer des requêtes d'un niveau d'abstraction plus fin.

- structurer la forme

Les commandes ayant été définies, il faut les présenter à l'utilisateur selon leur niveau de spécificité, de façon à ne pas embrouiller l'esprit de l'utilisateur avec un grand nombre de commandes hétérogènes mais plutôt à le guider naturellement vers le chemin de résolution de son problème.

Dans ODILE, la commande "Traiter..." concentre l'essentiel des fonctionnalités d'ODILE puisqu'elle permet le calcul d'un ensemble de clés potentielles à partir d'une chaîne de caractères et la recherche de tout ou partie de cet ensemble de clés dans le dictionnaire usuel actif.

Cependant, ODILE n'interdit pas à l'utilisateur de faire appel à la lemmatisation ou à l'outil dictionnaire de façon indépendante. L'utilisateur peut, s'il le désire, lemmatiser une chaîne puis consulter le résultat par la commande "Lemmatiseur", ou encore rechercher une clé dans le dictionnaire usuel par la commande "WinTool™...".

Du point de vue de la forme, le menu ODILE ne noie pas l'utilisateur sous le nombre des commandes disponibles. Il présente au contraire un nombre restreint de commandes et options facilement identifiables, qui correspondent à un contexte bien défini et dont l'activation provoque l'affichage à l'écran d'un certain nombre de commandes spécifiques à ce contexte.

* Flexibilité

- réparations lexicales

La flexibilité du lexique d'un système interactif est importante puisqu'elle permet sa modification sans avoir à modifier le logiciel source.

ODILE, comme toute application Macintosh, possède cette qualité. Cela est possible grâce aux fichiers ressources à la base de la programmation de logiciels sur Macintosh.

ODILE prévoit en outre la possibilité de modifier les équivalents claviers qui rentreraient en conflit avec ceux de l'application hôte. ODILE permet encore de choisir la visualisation ou la non visualisation de résultat de lemmatisation. Son interface est donc adaptable aux besoins de l'utilisateur. C'est une qualité appréciée car importante pour l'ergonomie.

2.3. Le modèle PAC et son application dans ODILE

2.3.1. Le modèle PAC

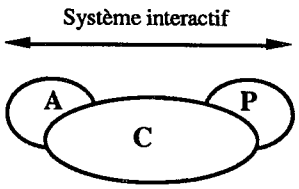
2.3.1.1. Définition

Selon le modèle PAC [COUT88], l'architecture d'un système interactif se décompose en trois parties qui sont la Présentation, l'Abstraction et le Contrôle :

- la Présentation définit l'Image du système, le comportement visible de celui-ci en réponse à des entrées de l'utilisateur ou à des sorties produites par le système.

- l'Abstraction désigne les fonctions du système.

- le Contrôle assure la communication entre Présentation et Abstraction.



La Présentation de PAC est réalisée par un ensemble d'agents spécialisés dans l'interaction avec l'utilisateur.

De la même façon que le système interactif se décompose en trois constituants, tout agent ou objet interactif se décompose en ces trois mêmes constituants:

- la Présentation de l'objet définit son comportement perceptible à l'utilisateur,

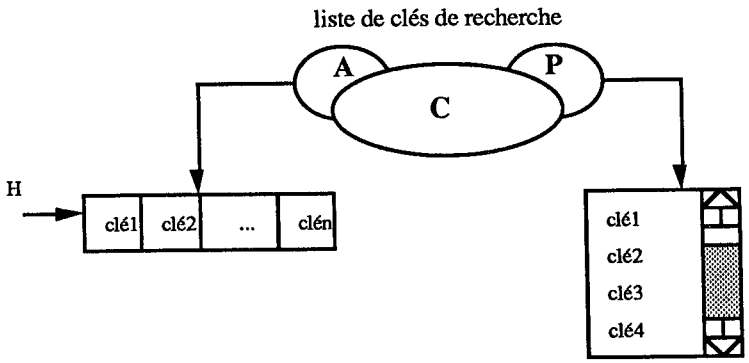
- l'Abstraction regroupe les fonctions et attributs fonctionnels visibles des constituants logiciels autres que la Présentation,

- le Contrôle gère les relations entre la Présentation de l'objet et son Abstraction.

Etudions l'objet interactif "liste de clés de recherche" utilisé notamment dans le dialogue engendré par la commande "Traiter..." d'ODILE.

- du côté présentation, la liste de clés est une liste statique affichée dans une fenêtre avec ascenseur, chaque clé figure sur une ligne différente.

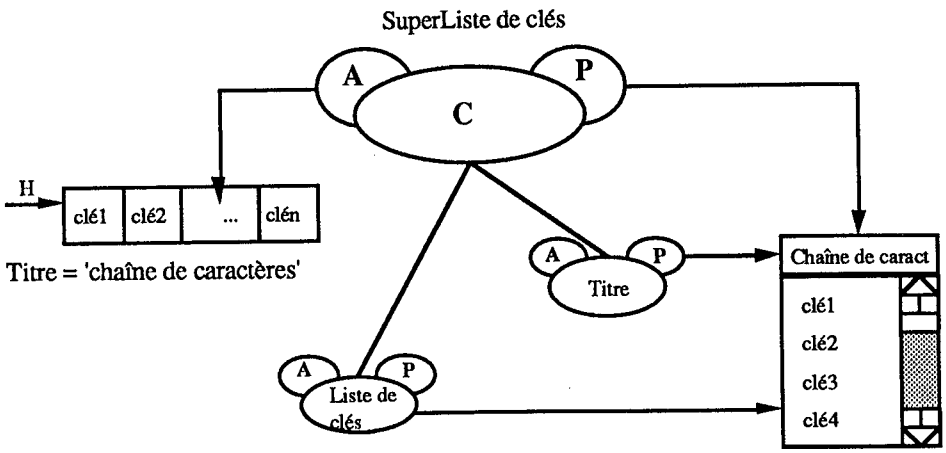
- du côté abstraction, cela peut être un tableau dynamique, référencé par un "handle", où chaque élément est une chaîne de caractères de longueur fixe.



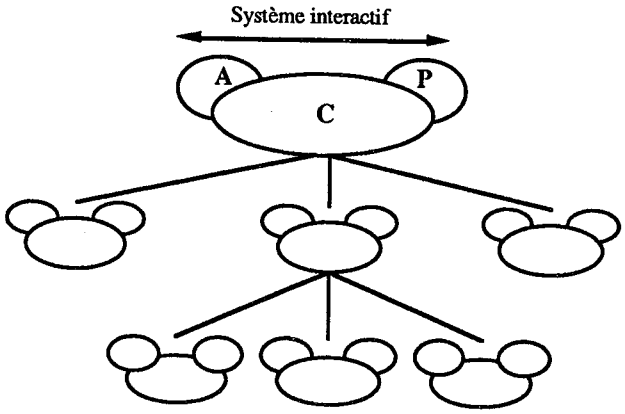
Les clés affichées à l'écran sont le reflet exact du contenu du tableau dynamique ; la suppression et l'ajout graphique d'une clé dans la liste doit provoquer la mise à jour de la structure interne correspondante et réciproquement. Cette cohérence entre abstraction et présentation est assurée par le contrôle.

Le modèle PAC définit la notion d'objet interactif composé. Le comportement d'un tel objet dépend de lui-même ainsi que des objets qui le constituent.

Reprenons l'exemple précédent. Supposons que le fenêtre présentant la liste des clés puisse être déplacée par l'utilisateur en cours de travail. Si l'on veut ajouter un titre à la liste, ce titre étant la chaîne de caractères à partir de laquelle les clés de la liste ont été calculées, la gestion des contraintes spatiales (titre au dessus de la liste) est réalisé par l'agent composé "SuperListe de clés".



Ainsi, par applications successives de la technique de raffinement, on peut structurer tout système interactif en une hiérarchie d'objets PAC.



Une des fonctions du contrôle de niveau le plus haut est de mettre en correspondance les concepts de l'application avec des abstractions d'objets interactifs. Il joue en quelque sorte le rôle de gare de triage. L'objet de niveau le plus haut est d'ailleurs appelé Super Contrôleur.
Une autre fonction du contrôle est de mettre en correspondance les actions de l'utilisateur avec la présentation des objets interactifs concernés.

2.3.1.2. Intérêt du modèle PAC

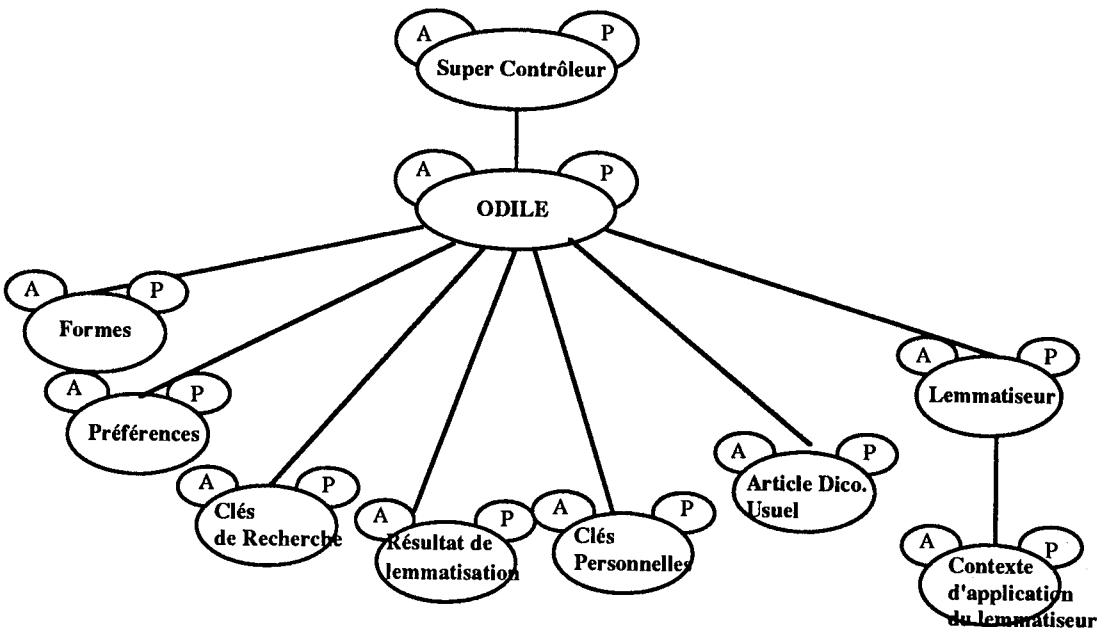
L'intérêt du modèle PAC est multiple. Il permet une *construction systématique* des systèmes interactifs par décompositions successives en objets PAC. La hiérarchie d'objets ainsi obtenus favorise une *évolution aisée de l'interface du système* puisque chaque objet présente deux facettes indépendantes, son abstraction et sa présentation, qui communiquent par un mécanisme de liaison, le contrôle, ce qui présente l'avantage de pouvoir en modifier une sans mettre en cause le fonctionnement de l'autre. La communication entre objets s'effectue entre objets hiérarchiquement dépendants par l'intermédiaire de leurs contrôles. L'état de l'interaction est donc géré par un ensemble de contrôles coopérants, ce qui autorise les *dialogues à plusieurs fils d'activité.*

Le contrôle d'un objet OBJ peut jouer le *rôle d'arbitre* pour les objets qui lui sont subordonnés. Il peut par exemple *gérer des contraintes spatiales* entre ces objets. Dans un environnement multitâche, il peut comporter des fonctions nécessaires à l'*allocation de ressources*, par exemple l'unité centrale : OBJ signale son besoin de l'unité centrale au contrôle de niveau le plus haut, qui l'alloue successivement aux interlocuteurs qui en ont fait la demande ; lorsque l'unité centrale est allouée à OBJ, celui-ci règle à son tour la priorité entre les demandeurs locaux. Le contrôle de l'objet OBJ peut encore *gérer la syntaxe distribuée*, lorsque la spécification d'une commande se traduit par un ensemble d'actions réparties sur des objets distincts de la Présentation.

En contrepartie de ces avantages, le modèle PAC présente tout de même quelques inconvénients et insuffisances. On peut signaler ici la *difficulté de conception d'un système interactif* selon ce modèle, même si une fois achevée celle-ci est souvent plutôt correcte. Les objets PAC et leurs relations ne sont pas toujours faciles à discerner, les règles préconisées par le modèle étant très générales. On peut noter également une insuffisance de PAC à l'heure actuelle dans la *modélisation de systèmes interactifs répartis*. PAC permet de construire l'architecture logicielle de tels systèmes, mais ne prévoit pas quel doit être le comportement de la présentation des objets PAC en cas de conflit graphique.

2.3.2. Architecture du système interactif dans ODILE

Nous avons mis en oeuvre le modèle PAC pour organiser le système interactif ODILE. Une première approche nous a permis de proposer l'organisation illustrée par le schéma qui suit.



Mais cette structure n'est pas satisfaisante pour plusieurs raisons liées à la spécificité de l'agent PAC super Contrôleur.

On remarque que la présentation du Super contrôleur est vide.
Le rôle de son contrôle est double dans le cas d'un système interactif sur Macintosh:

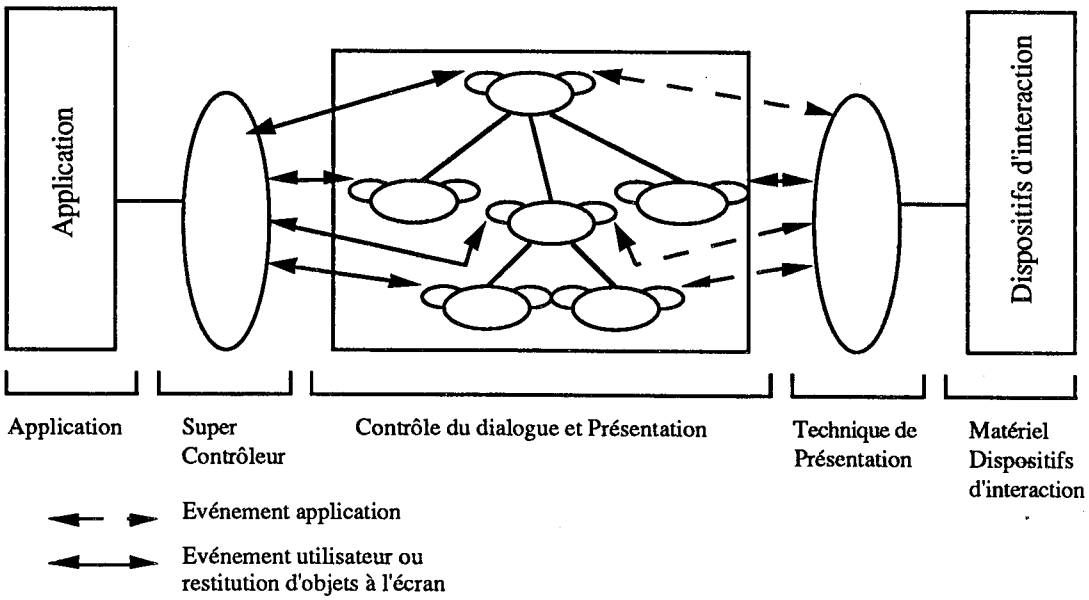
- le contrôle effectue bien sûr l'interfaçage entre le formalisme des abstractions des objets et le formalisme des fonctions de l'application. C'est un premier rôle du contrôle.
- La boîte à outils du Macintosh ne pratique pas le modèle du protocole embarqué, c'est-à-dire que le mécanisme de traitement des événements n'est pas intégré à l'entité de

dialogue (bouton, item de menu, fenêtre de dialogue, ...) ; l'entité de dialogue ne reçoit donc pas automatiquement les événements pour lesquels elle a de l'intérêt.
Un deuxième rôle du contrôle du Super Contrôleur est, dans ce cas, d'analyser la nature de l'événement reçu afin de le diriger vers l'objet concerné.

Si l'on examine les deux rôles du contrôle, nous pouvons faire plusieurs remarques:

- on constate en pratique que l'appel aux fonctions de l'application ne traverse pas la hiérarchie des objets PAC car, dans ce cas, aucune information n'est à propager. Les abstractions des objets de niveau inférieur communiquent directement avec l'abstraction de plus haut niveau, c'est-à-dire l'application.
- réserver au contrôle du Super Contrôleur le rôle de distribuer les événements vers les objets concernés risque de mettre en péril la portabilité du système interactif en le restreignant aux machines dont la boîte à outil ne pratique pas le modèle du protocole embarqué.

Ces constatations vont dans le sens de la définition d'une autre vue du modèle PAC où le Super Contrôleur n'est plus un agent PAC. Nouv-PAC est présenté par Laurence NIGAY dans [NIGAY90] Section I, chapitre 3.



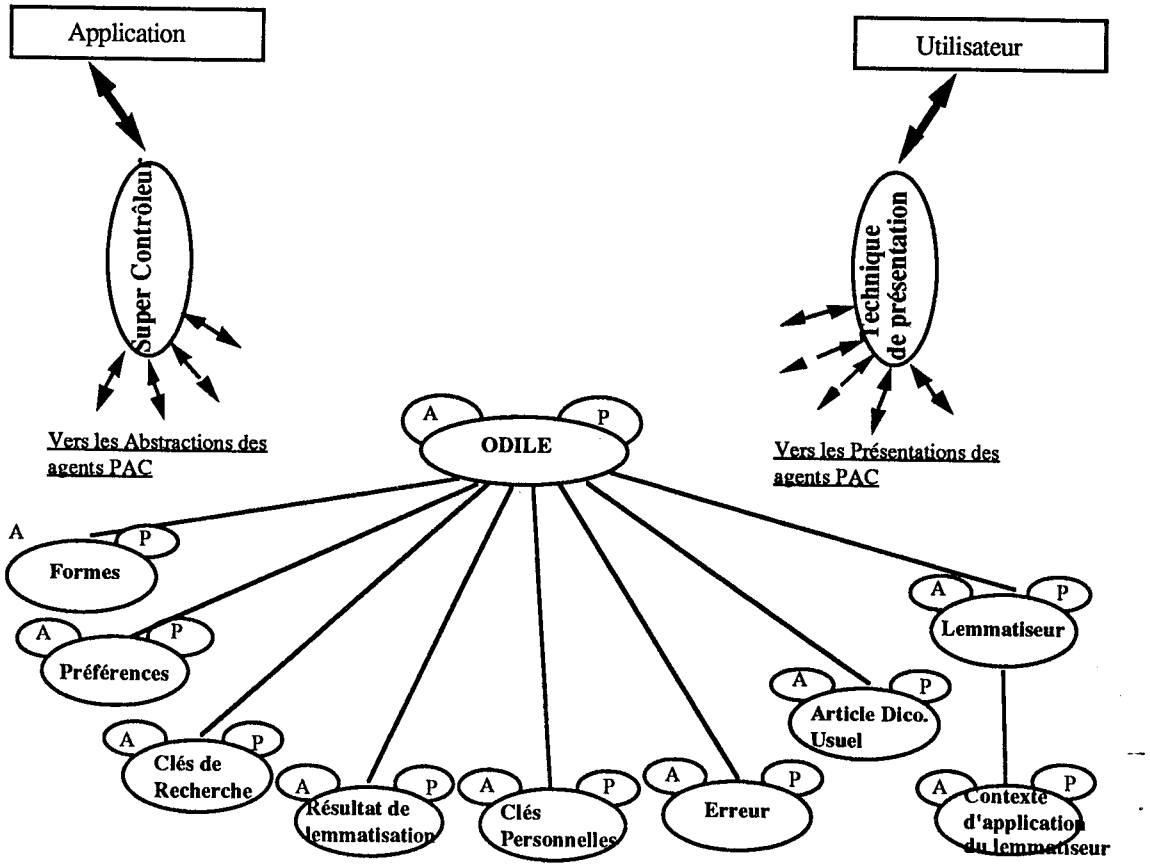
L'agent PAC Super Contrôleur a été éclaté.

Son abstraction possède les fonctions abstraites de l'application (lemmatisation, combinaison de formes et de lemmes, accès aux dictionnaires...) et devient le composant Application. Sa présentation est vide. Son contrôle est scindé en deux composants, le composant Technique de présentation et le composant Super Contrôleur.

Le composant Technique de Présentation effectue l'interface entre les expressions abstraites en provenance des présentations des objets PAC et les termes du lexique de sortie (primitives graphiques). Il possède les primitives de restitution des objets de niveau inférieur à l'écran et les primitives d'interprétation des actions de l'utilisateur (Routines de la ToolBox sur Macintosh). Dans le cas où ce composant utilise une boîte à outils telle que le mécanisme de traitement n'est pas intégré aux éléments de dialogue, ce composant doit répartir les événements reçus vers les objets concernés.

Le composant Super Contrôleur assure l'interface entre le composant Application et l'interface utilisateur nommée aussi Contrôle du dialogue et Présentation.

Le schéma suivant présente ODILE structuré selon le modèle Nouv-PAC.



Les abstractions des objets de niveau inférieur communiquent "directement" avec l'abstraction de plus haut niveau via une interface entre le système interactif et l'application appelée Super Contrôleur et qui adapte les formalismes des deux abstractions.

De même, la présentation de plus haut niveau communique "directement" avec les présentations des objets de niveau inférieur via une interface le système interactif et l'utilisateur appelée Technique de Présentation. Cette interface analyse les événements produits par l'action de l'utilisateur, détermine leur nature et les répartit vers la présentation des objets concernés. En

ce qui concerne la restitution à l'écran des objets de niveau inférieur, la présentation de ces objets appelle directement les primitives de la Technique de Présentation, dans notre cas des routines de la ToolBox du Macintosh.

Cette modification est intéressante dans la mesure où la présentation et l'abstraction des objets de niveau inférieur n'auront plus à "traverser" les contrôles des objets dont ils sont hiérarchiquement dépendants pour communiquer avec l'abstraction et la présentation du niveau le plus haut, et vice versa.

Remarquons que, sur ce schéma, l'objet Erreur est apparu. Cet agent est dédié à la gestion des messages d'erreur paramétrés. Une branche partant de l'objet Lemmatiseur se termine par trois petits points : cela représente les objets correspondant aux diverses données nécessaires au lemmatiseur et que l'on pourra gérer dans une version ultérieure d'ODILE.

Précisons maintenant le rôle de l'abstraction, de la présentation et du contrôle de chaque objet. Pour chaque objet, le contrôle a en charge le changement de formalisme entre l'abstraction et la présentation. Il assure aussi la communication avec les objets adjacents dans la hiérarchie.

ODILE

Son abstraction mémorise les options de traitement et les préférences désignées par l'utilisateur. Elle réinitialise les variables de l'application à chaque nouvelle demande de traitement.

Sa présentation affiche la barre des menus et gère les actions de l'utilisateur sur les articles de menu.

Formes

Son abstraction détermine l'ensemble des formes de la chaîne de caractères à analyser, et les mémorise au niveau de l'application.

Sa présentation déclenche un événement de copie de la chaîne sélectionnée dans le presse papier. Elle affiche le dialogue de saisie d'une chaîne de caractères si besoin est.

Préférences

Son abstraction mémorise l'ensemble des paramètres d'utilisation de l'accessoire de bureau, à savoir les équivalents claviers, le nombre maximum d'occurrences traitables, l'indicateur signifiant si l'accessoire doit afficher ou non les résultats intermédiaires de lemmatisation.

Sa présentation gère les actions de l'utilisateur dans le dialogue de paramètrage.

Clés de recherche

Son abstraction calcule, en fonction des options de traitement qui lui sont transmises par le contrôle d'ODILE, les clés potentielles de recherche à partir des formes qu'elle récupère au niveau de l'application (par appel aux procédures appropriées du module de lemmatisation). Elle mémorise ces clés. Elle mémorise, dans une liste au niveau de l'application, les dictionnaires usuels qu'elle ouvre (par appel aux procédures appropriées du module outil dictionnaire) et celui qui est actif. De même, dans une autre liste, elle garde les lexiques personnels qu'elle ouvre et celui qui est actif. Elle garde au niveau de sa propre mémoire le contenu de la clé sélectionnée. Elle exécute les actions demandées par l'utilisateur sur la liste des clés.

Sa présentation affiche la liste des clés calculées par l'abstraction. Elle présente sous forme de boutons les actions possibles sur cette liste de clés (Ajouter, Supprimer, Retraiter, Rechercher), de menus fugitifs les actions possibles sur les dictionnaires usuels et lexiques personnels (Installer). Elle gère a tout moment la cohérence des commandes disponibles avec le contexte.

Résultat de lemmatisation

Son abstraction mémorise la structure résultant de la lemmatisation faite par l'objet "Clés de recherche" et transmise par le contrôle d'ODILE.

Sa présentation est responsable de l'affichage graphique de cette structure de sortie.

Clés personnelles

Son abstraction constitue, dans sa mémoire propre, la liste des lexiques personnels installés (installés par l'objet "clés de recherche"). Elle obtient ces informations au niveau de l'application. Elle augmente cette liste de ceux qu'elle ouvre en supplément. Elle exécute les actions de gestion de ces fichiers. Elle met à jour la liste des lexiques ouverts dans sa mémoire propre et au niveau de l'application si elle ferme un lexique.

Sa présentation permet de lister à l'écran le contenu du lexique personnel actif. Elle présente sous forme de boutons les actions possibles sur les enregistrements et les fichiers de clés personnelles et prend en compte les actions de l'utilisateur.
En fait, l'abstraction et la présentation de cet objet sont en grande partie constituées par l'appel aux procédures appropriées de l'outil dictionnaire.

Erreur

Son abstraction a en charge la construction des messages d'erreur et, entre autres, l'évaluation des différents paramètres pour les construire. Elle mémorise la gravité de l'erreur. Celle-ci lui est transmise via ODILE par l'objet ayant détecté l'erreur.

Sa présentation affiche le message d'erreur suivant le modèle identifié par son contrôle.
Son contrôle gère le lien entre le niveau de gravité mémorisé dans l'abstraction et le modèle d'affichage du message d'erreur que doit générer la présentation.

Article Dico. Usuel

Son abstraction et sa présentation sont respectivement identiques à celles de l'objet "Clés personnelles" à la seule différence que là, ces composants travaillent sur les dictionnaires usuels.

Lemmatiseur

Son abstraction calcule les lemmes correspondant à la chaîne spécifiée dans une zone de saisie de sa présentation, par appel de la fonction appropriée du module lemmatiseur.

Sa présentation permet la saisie d'une chaîne à lemmatiser et assure l'affichage des lemmes calculés (et de certaines informations ayant servi au calcul). Elle présente sous forme de boutons les actions possibles sur les dictionnaires ou la grammaire du lemmatiseur courant, et prend en compte les actions de l'utilisateur.

Contexte d'application du lemmatiseur

Son abstraction effectue les opérations de gestion autorisées sur les éléments du dictionnaire et sur les données de la grammaire. Ces opérations s'appliquent au dictionnaire et à la grammaire utilisés par le contexte d'application courant du lemmatiseur, s'ils existent.

Sa présentation est responsable de l'affichage des dialogues à travers lesquels l'utilisateur va pouvoir spécifier les commandes à appliquer à telle ou telle base du dictionnaire ou à telle ou telle donnée de la grammaire. Ces dialogues, qui ne sont pas encore définis, sont spécifiques à chaque lemmatiseur. La présentation varie selon le lemmatiseur employé.

3. La communication dans ODILE

Un des souhaits à satisfaire dans la conception d'ODILE est, nous l'avons vu plus haut, de pouvoir intégrer n'importe quel lemmatiseur et n'importe quel outil dictionnaire.

Cela est certainement un peu irréaliste. Néanmoins, en définissant, d'une part, des structures standard d'entrée et de sortie des lemmatiseurs, et d'autre part, des structures standard d'entrée et de sortie des outils dictionnaire, nous pourrons atteindre notre objectif, à la condition que les candidats à l'intégration ne satisfaisant pas ces normes soient légèrement adaptés pour s'y conformer. Nous devons bien sûr également standardiser les procédures qui manipulent ces données.

Nous avons défini ces structures standard de façon à ce qu'elles correspondent au surensemble le plus petit possible des structures des lemmatiseurs et des outils dictionnaires que nous avons pu étudier. Nous avons ensuite défini des interfaces programmes standard visant à adapter leur formalisme pour leur manipulation par ODILE.

Dans ce qui suit, nous présentons les structures standard définies et les points d'entrée exploitables par ODILE pour chacun des types d'outils. Les paramètres d'entrée et de sortie de ces points d'entrée sont définis à partir des éléments communs aux structures d'entrée et de sortie des logiciels de la même famille, afin de garantir des structures de communication qui soient les plus universelles possibles. Enfin, nous étudions l'interface programme standard à mettre en place pour la communication de chaque type d'outil avec ODILE.

3.1. Interface avec les lemmatiseurs

3.1.1. Fonctionnalités des lemmatiseurs

Le rôle essentiel du lemmatiseur dans ODILE est le calcul de lemmes correspondant au groupe d'occurrences fourni en entrée. Ce calcul a été décrit plus haut (paragraphe 2.1.1. de ce chapitre). Certains lemmatiseurs permettent d'obtenir une sorte de trace d'exécution du lemmatiseur. C'est une autre fonctionnalité exploitable dans ODILE.

Par exemple, le lemmatiseur extrait de PILAF propose, en mode interactif, une trace en deux étapes. A l'issue de l'analyse morphologique du groupe d'occurrences fourni, il affiche pour chaque occurrence les différents modèles exploités pour son analyse et les différentes interprétations morphologiques déterminées. Une interprétation morphologique est identifiée par une classe lexicale et des variables morphologiques. A l'issue de la lemmatisation qui suit, il affiche pour chaque occurrence les lemmes factorisés selon les chaînes de caractères communes, en indiquant pour chaque lemme affiché toutes les interprétations morphologiques.

La mise en œuvre des fonctionnalités décrites se traduit par l'appel de deux points d'entrée :

- **procedure** Lem_ChargerDonnees;

Cette procédure, sans paramètre, charge en mémoire centrale les données nécessaires à l'analyse de la chaîne de caractères fournie en entrée, suivant le lemmatiseur utilisé,

- **procedure** Lem_Lemmatiser (Chaine; Affiche_visible: booleen; **var** StructAffich: Fichier Texte; **var** StructSortieLem) ;

La procédure Lem_Lemmatiser lemmatise la chaîne de caractères, Chaine, fournie en entrée et construit une structure StructAffich, si l'indicateur Affiche_visible est à Vrai. Dans tous les cas, cette procédure construit la structure des résultats de lemmatisation StructSortieLem.

Chaine est la chaîne de caractères à lemmatiser. Cette chaîne de caractères est une expression régulière de la forme : <chaîne sans blanc>blanc(<chaîne sans blanc>blanc)*. On dira alors que cette chaîne est normalisée.

Affiche_visible : Booleen contrôle la production de la structure StructAffich.

StructAffich : Fichier texte est une structure intermédiaire pour l'affichage des résultats de lemmatisation. Elle récapitule les modèles utilisés pour l'analyse de chaque occurrence de la chaîne d'entrée et les interprétations morphologiques déterminées correspondantes, puis pour chaque interprétation morphologique de chaque occurrence, les lemmes déterminés. Si l'on n'a pas besoin de cette structure, on met Affiche_visible à Faux.

StructSortieLem est la structure de sortie, construite dans tous les cas.

Le type de certains paramètres (Chaine en entrée, StructSortieLem en sortie) n'est pas déterminé. C'est l'objectif des paragraphes qui suivent.

3.1.2. Choix de la structure d'entrée des lemmatiseurs

La structure d'entrée d'un lemmatiseur est une chaîne de caractères. Suivant les cas, cette chaîne de caractères est traitée après normalisation (PILAF) ou bien après détermination des occurrences (ou formes) composées par cette chaîne (ATEF [CHAU72], analyseur de surface développé par P. PALMER [PALM90]). Nous conservons cette structure de chaîne, en la

limitant toutefois à 255 caractères, comme structure d'entrée commune à tout lemmatiseur intégrable à ODILE (paramètre `Chaine`).

3.1.3. Choix de la structure de sortie des lemmatiseurs

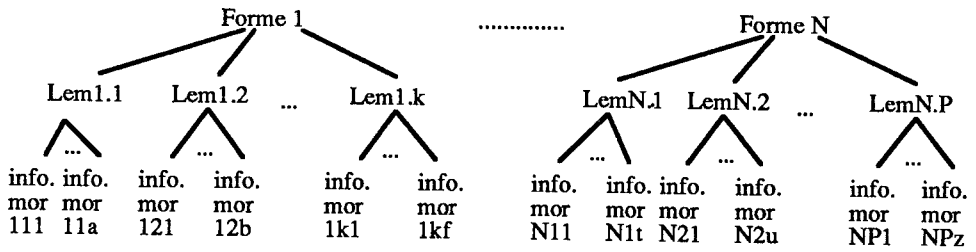
3.1.3.1. Choix de la structure

Nous allons étudier une structure de sortie possible en examinant celles qui nous sont proposées par le lemmatiseur issu de PILAF et l'analyseur morphologique ATEF.

Suite aux aménagements faits en juin 1989 [CHAP89b] sur le lemmatiseur PILAF, pour l'ensemble d'occurrences à analyser suivant:

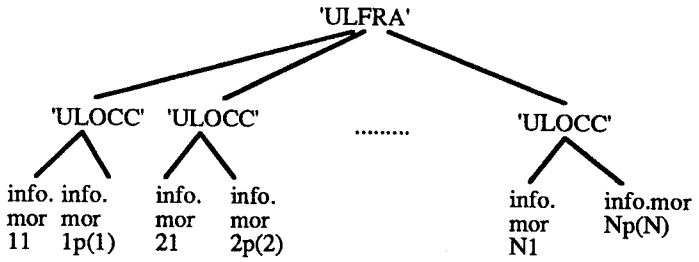
'Forme1 Forme2 ... Forme N'

nous avons la structure de sortie ci-dessous:

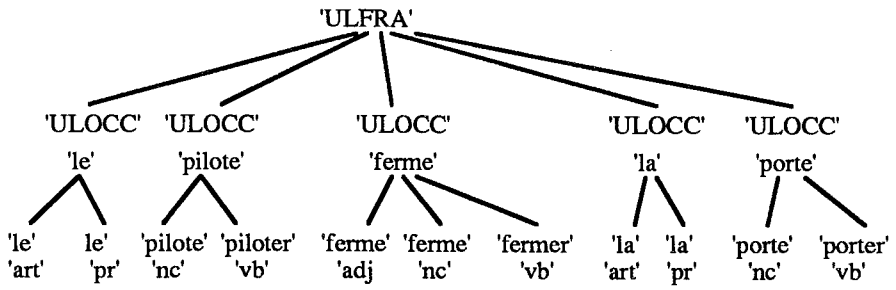


Le résultat de l'analyse est ici fourni séparément pour chaque forme.
Pour chaque forme d'origine, nous pouvons avoir plusieurs lemmes, chaque lemme pouvant être issu de plusieurs interprétations morphologiques.

Dans ATEF, la forme standard de sortie est une forme arborescente se présentant de la façon suivante:

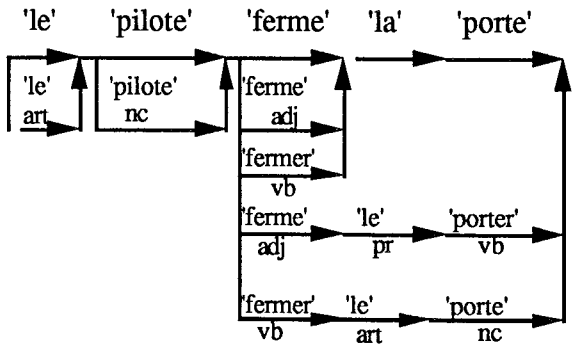


Sur l'exemple "Le pilote ferme la porte", cela donne le résultat suivant :

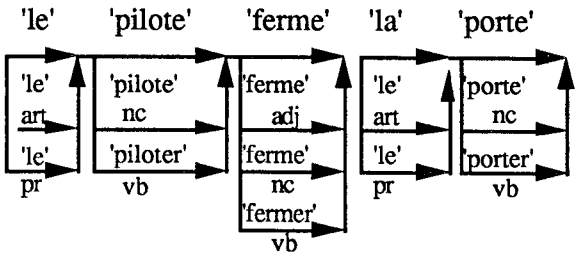


Dans les résultats présentés ci-dessus, rien ne nous est dit sur le façon de combiner les lemmes de chaque occurrence.

Une forme de résultat possible pour ATEF est une forme compatible avec les systèmes-Q. Il s'agit d'un graphe de chaînes qui rend compte des chemins possibles.
Examinons à nouveau l'ensemble d'occurrences suivant : "le pilote ferme la porte".
Avec ATEF on obtient le résultat :

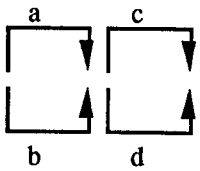


Avec PILAF, nous pouvons aisément obtenir un résultat sous la même forme, ce qui donnerait:



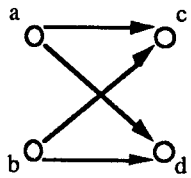
Sa forme est légèrement différente dans la mesure où PILAF n'effectue pas d'analyse de contexte au niveau morphologique, et n'élimine donc pas des solutions aberrantes (chemins passant par "piloter" vb ou "ferme" nc, ...). Dans PILAF, la vérification syntaxique est faite dans une étape suivante par construction d'un graphe de dépendance.

Considérons le graphe compatible avec les systèmes-Q suivant:



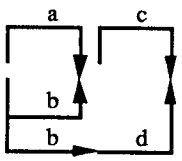
(1)

Ce graphe se traduit facilement par le graphe-avant (2), en remplaçant les arcs par des noeuds, et les transitions possibles par des arcs entre ces noeuds.



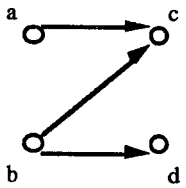
(2)

Si l'on souhaite interdire la combinaison "ad", cela se traduira par l'introduction d'une redondance dans (1):



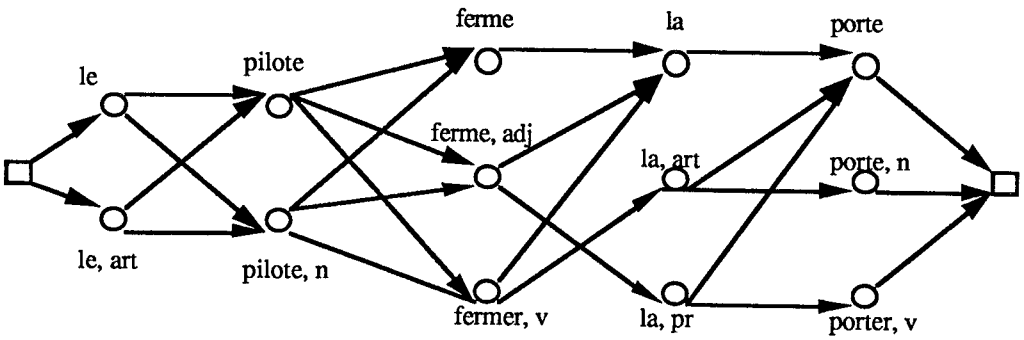
(1')

or dans (2), cela est très facile à exprimer:



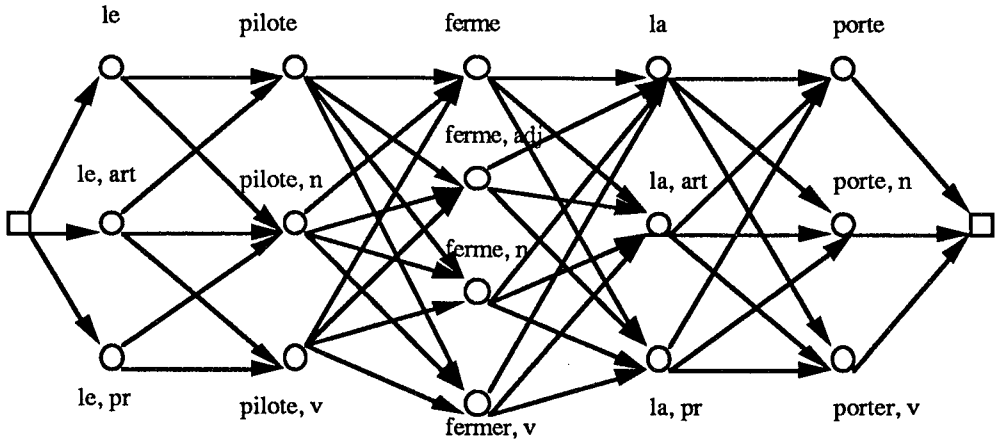
(2)

Dans ce formalisme, le résultat de l'analyse de la phrase "Le pilote ferme la porte" par l'analyseur ATEF, donne le graphe (3).



(3)

dans ce même formalisme, le graphe (4) peut être le résultat obtenu par PILAF.



(4)

Ce formalisme semble répondre à notre attente:

- il a une représentation simple.
- il met en évidence les combinaisons possibles, en éliminant les solutions aberrantes lorsque le lemmatiseur le permet.

C'est pourquoi nous choisissons le formalisme du graphe-avant comme structure de sortie commune à tout lemmatiseur candidat à l'intégration dans ODILE (paramètre `StructSortieLem`).

3.1.3.2. Informations fournies

La structure de sortie sous forme de graphe-avant étant choisie, reste à définir la nature des informations portées par les nœuds de ce graphe.

L'information fondamentale d'un nœud est une chaîne de caractères (`Valchaine`) pouvant représenter deux choses:

- soit une forme de la chaîne de caractères d'origine analysée,
- soit un des lemmes résultat de l'analyse de cette forme.

Nous aurons donc une information associée à cette chaîne, sa nature, c'est-à-dire 'F' pour une forme, 'L' pour un lemme.

Une autre information à associer à chaque chaîne `Valchaine` est son rang dans la chaîne de caractères d'origine. C'est le rang de l'occurrence, dans l'ensemble d'occurrences à analyser, dont `Valchaine` est un résultat. Cette information, avec l'information nature, sera utile dans une version ultérieure d'ODILE dans laquelle des options d'ordonnancement des clés potentielles pourront être proposées à l'utilisateur (cf paragraphe 4. Bouton "chercher" page 51)

Si l'on prend l'ensemble d'occurrences "chevaux de frise", on obtient huit noeuds (en supposant, bien sûr, que les dictionnaires utilisés pour l'analyse ne contiennent pas de tournures) ayant les caractéristiques suivantes:

CHAINE	NATURE	RANG	CATEGORIE
"chevaux"	'F'	1	nom
"cheval"	'L'	1	nom
"de"	'F'	2	indef
"de"	'L'	2	art
"de"	'L'	2	prep
"frise"	'F'	3	indef
"frise"	'L'	3	nom
"friser"	'L'	3	verbe

Supposons que notre outil supporte les options d'ordonnancement des clés. Examinons alors le résultat de la phase de calcul de toutes les combinaisons non redondantes de formes et lemmes sur l'ensemble des occurrences, avec l'option "FFFL" active. Cette option signifie "les formes sont prioritaires et les lemmes progressent par la droite". On obtient la liste des quatre clés de recherche potentielles suivantes:

"chevaux de frise" (FF)
"chevaux de friser" (FL)
"cheval de frise" (LF)
"cheval de friser" (LL)

Cependant l'information (origine, longueur) d'une occurrence résultat R, plutôt que rang, a l'avantage de résoudre le problème posé par certaines occurrences dont l'analyse fournit plusieurs chaînes résultat de longueur différente.

Il s'agit de l'origine, dans la chaîne originelle, de l'occurrence analysée. La longueur est le nombre de caractères de la sous-chaîne analysée.

Exemple :
soit la chaine d'entrée : "microordinateurs",
il est possible qu'un lemmatiseur détermine les éléments suivants :

Forme ou Lemme		(Origine, Longueur)	
"micro"		L	(0, 5)
"ordinateur"		L	(5,11)
"microordinateur"	F	(0,16)	

ODILE, qui recevra la structure de sortie du lemmatiseur sous forme de graphe-avant, pourra calculer les combinaisons de formes et de lemmes en utilisant l'information longueur ; en effet, à origine égale, ODILE choisira en priorité le nœud du graphe correspondant à la sous-chaîne la plus longue (ou la moins longue selon le choix fait) pour constituer les combinaisons.

Bien sûr, chaque noeud du graphe possédera également les informations issues directement de l'analyse morphologique, à savoir la catégorie et les variables morphologiques.

La structure de sortie n'est pas seulement destinée à satisfaire les besoins d'ODILE. Il doit s'agir d'une structure universelle, nous l'avons vu plus haut. En particulier, nous pouvons adjoindre à chaque noeud une information destinée à une analyse structurale éventuelle ultérieure. Cette information est la nature de la chaîne portée par le noeud, en référence à l'occurrence ou aux occurrences analysées.
Il s'agit d'un scalaire pouvant prendre la valeur :

- O pour indiquer que la chaîne correspondant au nœud constitue une occurrence dans la chaîne ; c'est une Occurrence complète,

- T pour indiquer que la chaîne recouvre plusieurs occurrences consécutives parmi les occurrences analysées ; c'est une Tournure.

- G, I, D pour indiquer que la chaîne en question recouvre une partie seulement de l'occurrence analysée ; c'est un morceau Gauche, Interne, ou Droit.

La structure de sortie est un graphe avant contenant les informations suivantes :

- Les nœuds représentent des formes ou des lemmes.
- Les arcs représentent les chemins possibles parmi ces nœuds.
- Chaque noeud contient jusqu'à dix (10) informations (entre parenthèses sont indiqués le nom de la variable et son type) :

1) le numéro du nœud (`Numéro : entier`) ;

2) son identification, obligatoire (`valChaîne : chaîne`) et ne contenant pas de blanc ;

3) son type obligatoire (`TypeChaîne`) : Forme ou Lemme (F | L) ;

4) sa nature, facultative (`Nature`) : Occurrence complète, Tournure, morceau Gauche, Interne, ou Droit (D | O | I | G | T) ;

5) sa localisation, obligatoire (`Origine : entier`), désigne l'origine (0 désignant la position précédent le premier caractère) de `ValChaine` ;

6) sa longueur, obligatoire (`Longueur : entier`), qui est le nombre de caractères couvert par le nœud (la longueur de la sous-chaîne traitée dans la chaine d'entrée) ;

7) sa catégorie morphologique (`Catégorie : chaîne`), facultative (par exemple : `(Catégorie 'adv)` ou `(Catégorie 'subc)` ou encore `(Catégorie 'dept))`;

8) ses variables morphologiques (`Variables : liste`), facultatives (par exemple : `(Variables('plus 'mas 'tre 'suj))` ou `(Variables((nombre 'plus) (genre 'mas) (fonction 'suj)))`;

9) des informations supplémentaires (`Reste : liste`), facultatives, définies suivant les besoins ;

10) les numéros des nœuds successeurs du nœud courant dans le graphe (`Suivant : liste d'entiers`), la liste vide (`()`) dénotant l'absence de successeur.

Pour une plus grande souplesse d'adaptation de divers lemmatiseurs à ODILE, le module de lemmatisation ne transmettra pas directement à ODILE la structure `StructSortieLem` dans le format interne manipulé par le lemmatiseur.

La structure transférée sera donc dans un format symbolique indépendant de la structure interne manipulée par le lemmatiseur et indépendant de la structure interne manipulée par ODILE. Nous avons choisi, pour ce faire, une structure de transfert sous forme d'un fichier texte "à-la-Lisp". Le fichier texte contient la suite des nœuds dans l'ordre ascendant des numéros. Chacun des nœuds est codé dans ce fichier comme suit :

```
(((Numéro Numéro) (ValChaîne ValChaîne)
  (TypeChaîne TypeChaîne) [(Nature Nature)]
  (Origine Origine) (Longueur Longueur)
  [(Catégorie Catégorie)] [(Variables
({Variables}+))]
  [(Reste ({Reste}+))])
(Suivants ({Suivant}+)
)
```

Notation : les caractères spéciaux { } [ ] * + sont utilisés pour coder des expressions régulières. Les parties entre crochets ([ ]) sont facultatives, les symboles en italiques dénotent les valeurs, et les accolades ({ }) permettent des regroupements. La construction $\{x\}+$ signifie

que x peut être répété une ou plusieurs fois, et $\{x\}^*$ signifie que x peut être répété zéro ou plusieurs fois. On pourra consulter un exemple de structure de sortie en annexe 2.

Le fichier texte étant une autre forme de la structure de sortie du module Lemmatiseur, la conversion "structure interne lemmatiseur - fichier texte" se fera dans une partie interface du module lemmatiseur.

3.1.4. Points d'entrée des lemmatiseurs

Nous récapitulons dans ce paragraphe les procédures correspondant aux points d'entrée de tout lemmatiseur intégrable à ODILE ainsi que leurs paramètres respectifs.

- **procedure** `Lem_ChargerDonnees;`
sans paramètres.

- **procedure** `Lem_Lemmatiser (Chaine: Chaine[255]; Affich_Visible: Booleen;` **var** `StructAffich: Fichier Texte;` **var** `StructSortieLem: Fichier Texte);`

Il est à noter que le préfixe "`Lem_`" est générique et est remplacé par un préfixe représentant le ou les lemmatiseurs utilisés (par exemple "`PL_`" pour PILAF).

3.1.5. Interfaçage Lemmatiseurs - ODILE

3.1.5.1. Paramètres d'entrée des lemmatiseurs

Dans ODILE, la chaîne de caractères à lemmatiser est soit selectionnée sous l'application d'où l'accessoire de bureau est appelé, soit saisie si aucune sélection n'a été trouvée. Afin d'éviter des calculs inutiles, la taille de la chaîne transmise au lemmatiseur peut être limitée. Cette limite, fixée par un paramètre d'ODILE, est modifiable par l'utilisateur et exprimée en nombre d'occurrences. Par défaut, le nombre maximum d'occurrences à analyser est fixé à 4. Mais cette limite n'est pas une contrainte. L'utilisateur averti d'un dépassement peut facilement la contourner et lemmatiser une suite d'occurrences aussi importante qu'il le souhaite.

Ce contrôle implique une évaluation du nombre d'occurrences de la chaîne à analyser avant de la transmettre au lemmatiseur intégré. En conséquence, la structure transmise par ODILE au lemmatiseur est constituée d'une poignée ("handle") référençant un tableau dynamique d'occurrences, HTabOcc, qui représente la chaîne à analyser, et du nombre d'éléments du tableau, NbOcc.

Exemple :
Soit N=4, le paramètre nombre d'occurrences maximum,
soit "Occ1 Occ2 Occ3 Occ4 Occ5 ... OccP", la chaîne de caractères à analyser,
alors, si l'utilisateur souhaite la limitation, ODILE transmet au lemmatiseur :

- le nombre d'occurrences NbOcc — NbOcc=4
- le tableau référencé par HTabOcc — HTabOcc^^[1] = "Occ1"
  HTabOcc^^[2] = "Occ2"
  HTabOcc^^[3] = "Occ3"
  HTabOcc^^[4] = "Occ4",

sinon, ODILE transmet au lemmatiseur :

- le nombre d'occurrences NbOcc — NbOcc=P
- le tableau référencé par HTabOcc — HTabOcc^^[1] = "Occ1"
  HTabOcc^^[2] = "Occ2"
  ...
  HTabOcc^^[P] = "OccP"

Nous pouvons remarquer que le contrôle du nombre maximum d'occurrences à analyser n'est exploitable que dans le cadre d'une mise en œuvre d'ODILE sur des langues possédant des séparateurs d'occurrences. Pour des langues n'en possédant pas, ODILE est néanmoins exploitable en fixant le paramètre N à 0. Cette valeur de N a pour signification de ne pas effectuer le découpage de la chaîne de caractères à lemmatiser en occurrences : la chaîne est alors considérée comme une seule occurrence, mémorisée dans un unique élément du tableau TabOcc.

La structure transmise par ODILE doit être convertie par une procédure de l'interface "Lemmatiseurs - ODILE", la procédure `I_Lem_ConstituerChaine`.

```
procedure I_Lem_ConstituerChaine (HTabOcc: TabOccHdl;
NbOcc: Entier; var Chaine: Chaine[255]);
```

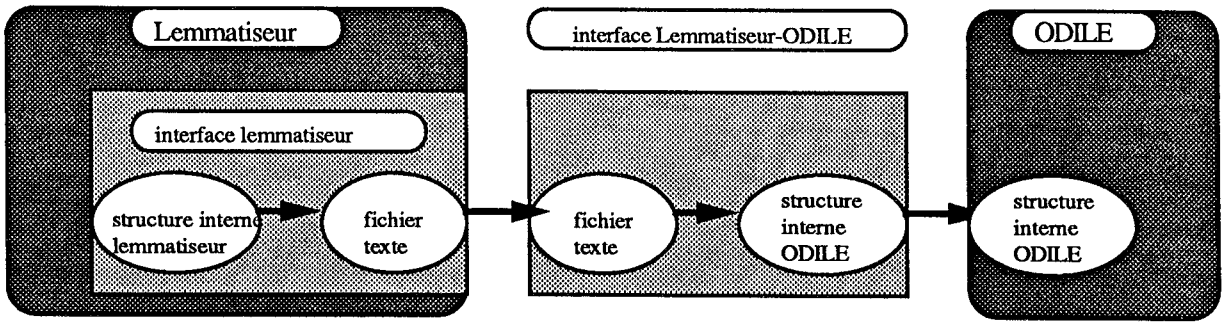
La procédure `I_Lem_ConstituerChaine` reconstitue à partir des `NbOcc` éléments du tableau `TabOcc`, la structure d'entrée commune attendue pour les lemmatiseurs intégrables, à savoir une chaîne de 255 caractères normalisée (cf paragraphe 3.1.1 page 72).

3.1.5.2. Paramètres de sortie des lemmatiseurs

Nous avons vu que tout lemmatiseur intégrable à ODILE fournit une structure de résultats de lemmatisation sous forme d'un fichier texte. Or, pour pouvoir être exploité par ODILE, elle doit être chargée dans une structure interne spécifique.

Cette structure spécifique est un tableau dynamique dont chaque élément regroupe tout ou partie des informations d'un nœud du graphe. Ce tableau est référencé par une poignée.

La conversion "fichier texte - structure interne ODILE" est réalisée par l'interface programme entre le module Lemmatiseur et ODILE et apparaît dans le schéma suivant :



Cette conversion est réalisée par la procédure suivante:

```
procedure I_Lem_ChargerStructure (NomFtext:  Chaine[255];
var HTabStructOdile: TabStructOdileHdl; var NbNoeud: Entier);
```

Il est à noter que la structure interne manipulée par ODILE (tableau dynamique référencé par la poignée `HTabStructOdile`) sera certainement moins riche que celle qui est fournie par le lemmatiseur. Dans un premier temps, par exemple, nous ne conserverons pas les informations Nature, catégorie et variables morphologiques.

3.2. Interface avec les outils dictionnaire

3.2.1. Fonctionnalités des outils dictionnaire

Pour pouvoir être intégré à ODILE, l'outil dictionnaire candidat doit posséder un ensemble minimal de fonctionnalités, telles que création de dictionnaires, gestion de ces dictionnaires, et bien sûr recherche dans ces dictionnaires.

Cependant, la mise en œuvre de certaines fonctionnalités peut diffèrer d'un outil dictionnaire à l'autre ; nous avons vu notamment que le format des dictionnaires et plus généralement des fichiers à gérer peut changer.

Nous ne nous intéressons pour l'instant qu'aux outils dictionnaires traitant une seule clé à la fois et gérant des fichiers d'enregistrements au format unique de type "clé + contenu". Pour étudier la communication d'ODILE avec cette classe d'outils dictionnaire, nous nous basons sur le modèle de fonctionnement de WinTool™.

3.2.2. Points d'entrée de WinTool™

Pour son intégration à ODILE, WinTool™ requiert les 7 points d'entrée suivants:

* Traitement de la commande d'ouverture de fichier des "pop-up menus" "Dico. Usuel" et "Lexique Perso."

Fonction `WT_Ouvrir (NomBase: Chaine[255]; VRefNum: Entier; TypeBase: Chaine[4]; Derrière: Pointeur): Entier;`

- elle ouvre le fichier (ou "base") WinTool™ identifié par `NomBase` et `VRefNum` et de type `TypeBase` (dictionnaire usuel ou lexique personnel),
- elle ouvre également la fenêtre WinTool™ associée à ce fichier en arrière-plan et plus précisément derrière la fenêtre dont le pointeur est spécifié en paramètre, `Derrière`.
- elle rend un entier positif qui est l'identificateur du fichier ouvert si l'ouverture s'est bien passée, un entier négatif sinon.

* Traitement du bouton "Chercher" du dialogue issu de l'article "Traiter..." d'ODILE

procedure `WT_Chercher (Cle: Chaine[60]; Numbase: Entier);`

- elle effectue la recherche de `Cle` dans le fichier WinTool™ identifié par `Numbase`,
- elle active la fenêtre WinTool™ associée au fichier identifié par `Numbase` pour la présentation du résultat de la recherche.

Le paramètre `cle` est une chaîne de 60 caractères puisque c'est une limitation actuelle de WinTool™.

* Traitement du bouton ">>" (Mémoriser) du dialogue issu de l'article "TRAITER" d'ODILE

Fonction `WT_Memoriser (Cle: Chaine[60]; Contenu: Poignée; NumBase: Entier) : Entier;`

- elle enregistre l'élément constitué par `Cle` et `Contenu` dans la base identifiée par `NumBase`,
- elle rend un entier qui spécifie si l'opération s'est bien passée : 0 --> OK, < 0 --> problème.

De même ici, le paramètre `cle` est limité à 60 caractères par WinTool™.

* Traitement de l'article "Gérer les clés personnelles..." d'ODILE :

```
Fonction WT-GererBase (NumBase: Entier): Entier;
```

- elle active la fenêtre associée au fichier WinTool™ d'identificateur `NumBase` ouvert,
- elle rend un entier qui spécifie si l'opération s'est bien passée : 0 --> OK, < 0 --> problème.

* Traitement de l'article de menu "WinTool™..." d'ODILE :

```
procedure WT_OpenWinTool (TheDCE: DCtlPtr);
```

Cette procédure joue le rôle de l'entrée Open de l'accessoire de bureau WinTool™ : elle effectue les initialisations nécessaires à l'exécution des procédures de WinTool™.
`TheDCE` est la zone de communication pour un accessoire de bureau. Elle a un type prédéfini dans la ToolBox du Macintosh, `DCtlPtr`.

* Traitement de l'article de menu "Fermer" d'ODILE :

```
procedure WT_CloseWinTool (TheDCE: DCtlPtr);
```

Cette procédure joue le rôle de l'entrée "Close" de l'accessoire de bureau WinTool™ : elle ferme tous les fichiers ouverts ainsi que toutes les fenêtres associées.

* Traitement d'une action de l'utilisateur dans une fenêtre générée par une procédure de WinTool™:

```
procedure WT_ControlWinTool(PB: parmBlkPtr; Device: DCtlEntry);
```

Cette procédure joue le rôle de l'entrée "Control" de l'accessoire de bureau WinTool™ : elle détermine la nature de l'événement et appelle la procédure de traitement de WinTool™ appropriée.

Il est à noter qu'au retour de tout appel d'un point d'entrée de WinTool™ rendant un entier, ODILE teste la valeur de cet entier. S'il est négatif (l'opération demandée s'est mal passée), le système interactif ODILE affiche un message indiquant la nature de l'erreur.

3.2.3. Interfaçage outils dictionnaire - ODILE

Quelques paramètres fournis par ODILE à l'appel d'un point d'entrée de WinTool™ ont des types différents de ceux attendus par WinTool™.

ODILE souhaite rechercher ou mémoriser dans des fichiers WinTool™ des clés de type chaîne de 255 caractères alors que les points d'entrée de WinTool™ (`WT_Chercher` et `WT_Memoriser`) n'acceptent que des clés de type chaîne de 60 caractères.

Le paramètre `CleOdile` est donc converti par la procédure d'interface suivante :

```
procedure I_WT_TronquerCle (CleOdile: Chaine[255];
var CleWT: Chaine[60]);
```

ODILE mémorise dans des fichiers WinTool™ des éléments "Clé + Contenu". Or, l'information Contenu fournie par ODILE est de type `TabCleHdl`, qui est une poignée sur un tableau dynamique d'au maximum 25 chaînes de 255 caractères, alors que le point d'entrée de WinTool™ (`WT_Memoriser`) n'accepte qu'un contenu de type Macintosh prédéfini `Handle` sur un tableau de caractères.

Le paramètre `Contenu` est converti par la procédure d'interface suivante :

```
procedure I_WT_TranscoderContenu (ContenuOdile: TabCleHdl;
var ContenuWT: Poignée);
```

L'utilisateur peut transmettre tout ou partie du contenu de l'enregistrement courant à l'application hôte par l'intermédiaire du presse-papiers en sélectionnant le texte correspondant avec la souris puis en lançant la commande "Copier". Ce mécanisme fonctionne si l'application hôte accepte la commande "Copier". Si c'est le cas, il est géré par un routine système du Macintosh. On voit donc que ce mécanisme ne nécessite aucune procédure de traitement de l'interface "ODILE - outil dictionnaire" supplémentaire.

4. Structure des dictionnaires

Une idée clairement définie au départ du projet était de modifier la structure de données des dictionnaires de lemmatisation. Ils n'étaient pas facilement modifiables et occupaient une place importante sur disque tout en proposant un nombre d'entrées pourtant encore insuffisant (cf paragraphe 2.1.2.1. du chapitre I).

Compte tenu de la stratégie linguistique utilisée, un nombre d'entrées correct se situe aux alentours de 100000. Il se trouve que l'équipe TRILAN du LGI a progressivement constitué un dictionnaire de 35000 entrées. Un tel dictionnaire permet de générer environs 250000 formes du français. Il traite environ 98% du français courant, le reste étant constitué de mots inconnus ou de formes appartenant à des domaines particuliers pouvant être traités par des dictionnaires terminologiques. C'est ce dictionnaire de 35000 entrées que nous allons exploiter dans l'outil final, et qui, dans une version suivante de l'outil, pourra être augmenté par l'utilisateur au fur et à mesure des utilisations. Il est donc impératif de revoir son organisation.

4.1. Quelques exemples d'organisation de fichiers

Il nous a paru intéressant d'étudier un certain nombre d'exemples d'organisation de fichiers. Dans un premier temps, nous présentons celle des fichiers de WinTool™ afin d'en définir une pour les lexiques personnels d'ODILE. Dans un deuxième temps, nous étudions celle des dictionnaires de différents lemmatiseurs afin d'ébaucher, pour PILAF, une organisation des dictionnaires de lemmatisation qui réponde mieux à nos besoins.

4.1.1. Dictionnaires WinTool™ et organisation en B-arbres

La gestion des fichiers manipulés par WinTool™ est effectuée par un gestionnaire de fichiers indépendant, WinTree™. C'est un gestionnaire de fichiers en séquentiel indexé.

Les "bases WinTool™" sont donc constituées de deux fichiers sur disque:

- le fichier des données,
- le fichier des index, qui est un arbre équilibré (Balanced-Tree ou B-Tree).

Selon la définition donnée dans [KNUT73] page 473, un B-tree d'ordre m est un arbre qui satisfait aux propriétés suivantes :

- chaque nœud a au plus m fils,
- chaque nœud, sauf la racine et les feuilles, a au moins m/2 fils,
- la racine a au moins 2 fils (sauf si c'est une feuille),
- toutes les feuilles sont au même niveau, et ne portent pas d'information,
- un nœud qui n'est pas une feuille et qui a k fils contient k - 1 clés.

Un arbre équilibré contenant N noeuds garantit que les longueurs de toutes ses branches sont du même ordre de grandeur ($\log_2 N$), et donc que le temps de recherche d'un article est, dans le pire des cas, égal à une valeur maximum.

Un B-tree d'ordre m > 2 est un arbre n-aire. Pour les fichiers stockés sur disque, les arbres n-aires sont plus intéressants que les arbres binaires car ils permettent, en particulier lors de la recherche d'une clé, d'avoir un nombre d'accès disque moins important.

Considérons le cas d'une base de 1000 articles. Le fichier des index contient donc 1000 index. Si, pour représenter ces index, nous prenons un arbre binaire l'arbre aura une profondeur de 10 ($2^9 < 1000 < 2^{10}$). La recherche d'un article nécessite au pire 10 accès disque. Par contre, si nous prenons un arbre n-aire dont chaque nœud contient 10 index, l'arbre aura une profondeur de 3 ($10^3 = 1000$). La recherche d'un article demandera au pire 3 accès disque. Bien sûr, le

nombre de comparaisons est plus important dans le deuxième cas, mais cela est négligeable puisque les opérations en mémoire centrale sont beaucoup plus rapides que les accès disque.

L'arbre des index définit une relation d'ordre sur les index. Dans notre cas, il s'agit de la relation "plus-petit que" lorsqu'on parcourt l'arbre de manière infixée.

La recherche d'une clé dans un B-tree d'ordre m de N clés demande un nombre d'accès h tel que $h \leq 1 + \log_{m/2} ((N + 1)/2)$. Si l'on considère N = 1000 et m = 10 la recherche d'une clé demandera au plus h = 5 accès disque.

L'insertion d'une clé dans un B-tree d'ordre m dont toutes les feuilles sont au niveau h a lieu dans un nœud de niveau h - 1. Si après insertion ce nœud contient m clés, on le divise en deux nœuds et on insère la clé médiane dans le nœud père du nœud originel. Ce processus se reproduit éventuellement jusqu'à éclatement de la racine de l'arbre. De cette façon l'arbre conserve ses propriétés et, en particulier, il reste équilibré.

On pourra consulter la description du mécanisme de recherche et d'insertion dans [KNUT73] pages 475-476.

4.1.2. Dictionnaire de clés personnelles

La structure externe des dictionnaires de clés personnelles est identique a celle des dictionnaires de WinTool™ : chaque élément est constitué d'une clé et d'un contenu. Mais on peut supposer que la taille de ces dictionnaires sera beaucoup plus modeste. Ils doivent en effet servir à répertorier les clés de recherche potentielles pour certaines suites d'occurrences pour lesquelles il est particulièrement difficile de constituer des entrées de dictionnaire usuel. De ce fait, on peut imaginer un simple fichier trié dans l'ordre lexicographique croissant des clés et un accès par dichotomie.

Néammoins, dans notre prototype, nous avons choisi de bénéficier des procédures de gestion de fichiers offertes par WinTree™, le gestionnaire utilisé par WinTool™. Nous avons donc opté pour une structure interne identique à celle des dictionnaires de WinTool™.

On peut noter ici que ce choix est contestable puisqu'il impose une légère modification du code de WinTool™ pour la prise en compte de ce type de fichier et que cela n'est pas conforme aux contraintes énoncées au départ du projet. Il conviendra donc de réexaminer ce choix pour des versions opérationnelles ultérieures.

4.1.3. Dictionnaires de lemmatisation

Nous présentons, dans les paragraphes qui suivent, les améliorations apportées à la structure du dictionnaire originelle, puis nous ouvrons une nouvelle voie de réflexion en proposant une organisation du dictionnaire radicalement différente.

4.1.3.1. Le dictionnaire PILAF

Le dictionnaire PILAF existe en fait en deux versions. La première est celle qui a été définie à l'origine et qu'ont exploité les participants au "projet MacPILAF" dont j'ai repris les travaux. Puis une autre version, visant essentiellement à réduire l'encombrement mémoire, a été étudiée et partiellement mise en œuvre en 1989 et 1990. Ces travaux ont été réalisés par Françoise BAINIER puis repris par Eric LOPEZ, deux étudiants du département informatique de l'IUT II de Grenoble dans le cadre de leur stage de fin d'étude au sein de l'équipe TRILAN.

Présentons tout d'abord la structure originelle du dictionnaire PILAF. Il contient un ensemble de bases de la langue française. Ces bases sont au cœur du traitement morphologique réalisé par PILAF.

INDECOUPABLE	MODELE	LETTRES	OCCLETTRES	CHCIRC	OCCCIRC
indique si la base est indécoupable ou non.	numéro de référence du modèle associé à la base.	chaîne de caractères donnant la base.	nombre d'occurrences de la base dans le dictionnaire.	chaînage circulaire de la base.	numéro d'occurrence de la base dans le chainage circulaire.
booléen 1 octet	**entier** 2 octets	**tableau de car.** 25 octets	**entier** 2 octets	**tableau de car.** 25 octets	**entier** 2 octets

LETTRES est le champ destiné à mémoriser la base. Sa taille est fixe quelle que soit la base. Il occupe 25 octets.

MODELE régit le comportement grammatical de la base en associant à la base l'ensemble des règles qui lui sont applicables. Toutes les bases ayant le même comportement ont le même modèle, c'est à dire le même numéro de référence au fichier des modèles dans le champ MODELE.

OCCLETTRES contient le numéro d'occurrence de la base LETTRES dans le dictionnaire. Ce champ sert à distinguer les bases alphabétiquement égales mais dont le comportement grammatical est différent.

Exemple :	LETTRES	OCCLETTRES	
	aliment	(2)	--> base du nom "aliment"
	aliment	(1)	--> base du verbe "alimenter"

Nous n'avons pas les même règles applicables pour les deux bases.

Le champ INDECOUPABLE d'une base est à faux si la base contient une sous-chaîne, qui est elle-même une base du dictionnaire, et dont le premier caractère coïncide avec celui de la base. Une condition supplémentaire limite les découpages : en fait, une base b est découpable en une base b' si, de plus, la génération de formes à partir de la base b' peut, entre autres, générer une forme dont un préfixe est égal à la base b.

Exemple :	LETTRES	OCCLETTRES	INDECOUPABLE
	couvent	(1)	F
	couv	(1)	V

"couv" ne peut pas être sous-découpée en "cou". En effet, "couv" peut générer "couvent" (verbe "couver" 3ème pers. présent indicatif ou subjonctif), qui est un préfixe (particulier) de la base "couvent", mais "cou" ne peut pas générer de forme préfixées par "couv" et encore moins par "couvent".

Les champs CHCIRC et OCCCIRC sont utiles à la génération, et bien sûr aussi pour la lemmatisation qui en est un cas particulier. Ces informations construisent un chaînage circulaire reliant toutes les bases intervenant dans la fabrication des formes fléchies d'une occurrence.

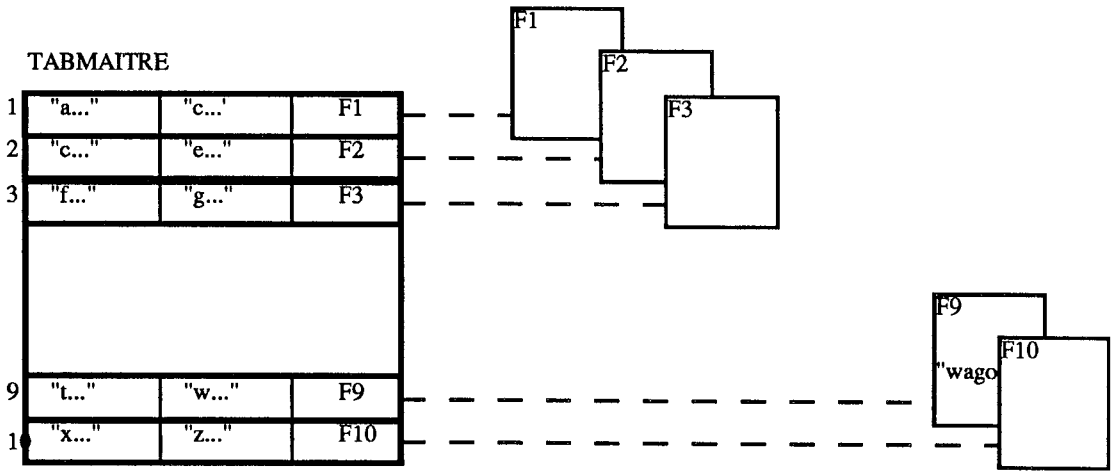
Exemple :
Le verbe "aller" fait intervenir 8 bases pour la génération de toutes ses formes fléchies, c'est-à-dire pour sa conjugaison.

LETTRES	OCCLETTRES	CHCIRC	OCCCIRC
"all"	(1)	"aill"	(1)
"aill"	(1)	"ir"	(1)
"ir"	(1)	"vont"	(1)
"vont"	(1)	"va"	(2)
"va"	(2)	"va"	(1)
"va"	(1)	"vas"	(1)
"vas"	(1)	"vais"	(1)
"vais"	(1)	"all"	(1)

Le fichier des bases est trié par ordre alphabétique décroissant pour les besoins de l'algorithme d'analyse et plus particulièrement pour la segmentation (recherche de segment identique, pour OCCLETTRES différent de (1), ou plus petit, pour INDECOUPABLE à faux, dans la suite du fichier des bases.)

Pour accélérer la recherche d'un segment dans le fichier des bases, celui-ci a été découpé en plusieurs fichiers. Le fichier TABMAITRE répertorie la première et la dernière base de chaque fichier. A l'initialisation, ce fichier est chargé dans une table en mémoire centrale. Il permet par comparaisons (opérations rapides en mémoire centrale) de déterminer dans quel fichier de bases le segment analysé se trouve si c'est une base. Pour trouver la base, on aura alors relativement peu d'accès disque à faire (recherche dichotomique dans le fichier identifié).

Prenons l'exemple d'un dictionnaire de 6000 bases. Le fichier TABMAITRE chargé en mémoire centrale aura dans ce cas 10 enregistrements. Nous aurons 10 fichiers de 600 bases.



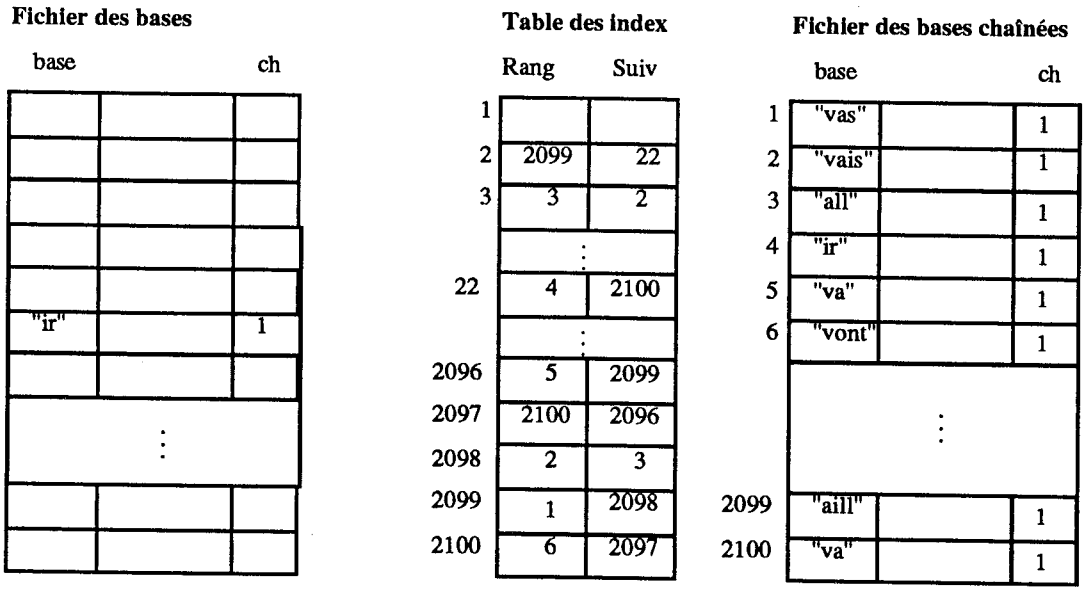
Si l'occurrence à analyser est "wagon", on aura 5 comparaisons (dichotomie) puis 10 accès disques ($2^9 < 600 < 2^{10}$), alors que sans cette organisation inspirée de la technique des B-arbres on aurait 13 accès disques ($2^{12} < 6000 < 2^{13}$). Cette structure permet une économie du nombre d'accès disque.
L'amélioration est d'autant plus significative que le nombre d'éléments du dictionnaire est grand, c'est-à-dire que le nombre de fichiers est grand. En effet, si l'on considère un dictionnaire de 35000 bases organisé en 350 fichiers de 100 éléments, le nombre d'accès disque moyen pour trouver une base est de 7 ($2^6 < 100 < 2^7$), alors que sans cette organisation il serait de 16 ($2^{15} < 6000 < 2^{16}$). Il faut tout de même garder un nombre raisonnable de fichiers pour que le fichier TABMAITRE puisse être chargé en mémoire centrale.

Cette structure reste néammoins très volumineuse puisqu'un dictionnaire de 35000 entrées occupe 1995000 octets (environ 2 Mo).

Une première idée a été mise en œuvre en 1989 par Françoise BAINIER. L'accès associatif à l'élément suivant du chaînage circulaire a été remplacé par un accès direct:. Autrement dit, la base suivante du chaînage a été remplacée par son adresse (n° fichier + rang

dans ce fichier). Le gain de place est non négligeable dans la mesure où, pour chaque entrée du dictionnaire on gagne 21 octets (25 octets de la base - 4 octets des deux entiers pour l'adresse), soit au total 735000 octets, c'est à dire environ 37% du volume de la base. Cette modification contribue à une rapidité plus importante de la génération en raison de l'accès direct à la base suivante dans le chaînage. Cependant, cette modification fige le dictionnaire. L'insertion ou la suppression d'une base impose de recalculer tous les accès directs : la gestion du dictionnaire est trop coûteuse. Cette solution a donc été abandonnée.

La deuxième idée, étudiée par Eric LOPEZ, a consisté à extraire du dictionnaire les informations liées au chaînage circulaire. Désormais, le fichier FCHAINE regroupe les 2100 bases environ possédant un chaînage circulaire. Les éléments de ce fichier sont référencés par une table d'index triée selon l'ordre alphabétique croissant des chaînes. Un booléen "CH" dans le fichier des bases indique, pour chaque base, si celle-ci possède un chaînage circulaire.



Le champ Rang de la table des index représente le rang d'une base dans le fichier des bases chaînées. Le champ Suiv est le rang, dans la table des index, de l'élément suivant du chaînage circulaire.

La recherche d'une base possédant un chaînage circulaire, dans le fichier des bases chaînées, est réalisée par dichotomie sur les chaînes de caractères via la table des index, car c'est elle qui est triée. Le parcours du chaînage est possible par accès direct à l'index de rang Suiv dans la table des index. L'accès direct aux éléments du chaînage circulaire favorise l'accélération des traitements de génération.

Le fichier des bases et celui des bases chaînées ont la même structure d'enregistrement, pour pouvoir utiliser certaines procédures communes de lecture des informations d'un enregistrement.

Bien que ceci ne soit pas conseillé en raison de la complexité des procédures de décodification des enregistrements, un gain de place supplémentaire (105 Ko) a été obtenu en codifiant tous les champs autres que le champ LETTRES sur un masque de bits de deux octets (au lieu de 5 octets auparavant):

bit 15 -> INDECOUPABLE (1=oui, 0=non),
bit 14 -> CH (1=oui, 0=non),
bits 13 à 11 -> OCCLETTRES (maximum 7);
bits 10 à 0 -> MODELE (maximum 440);

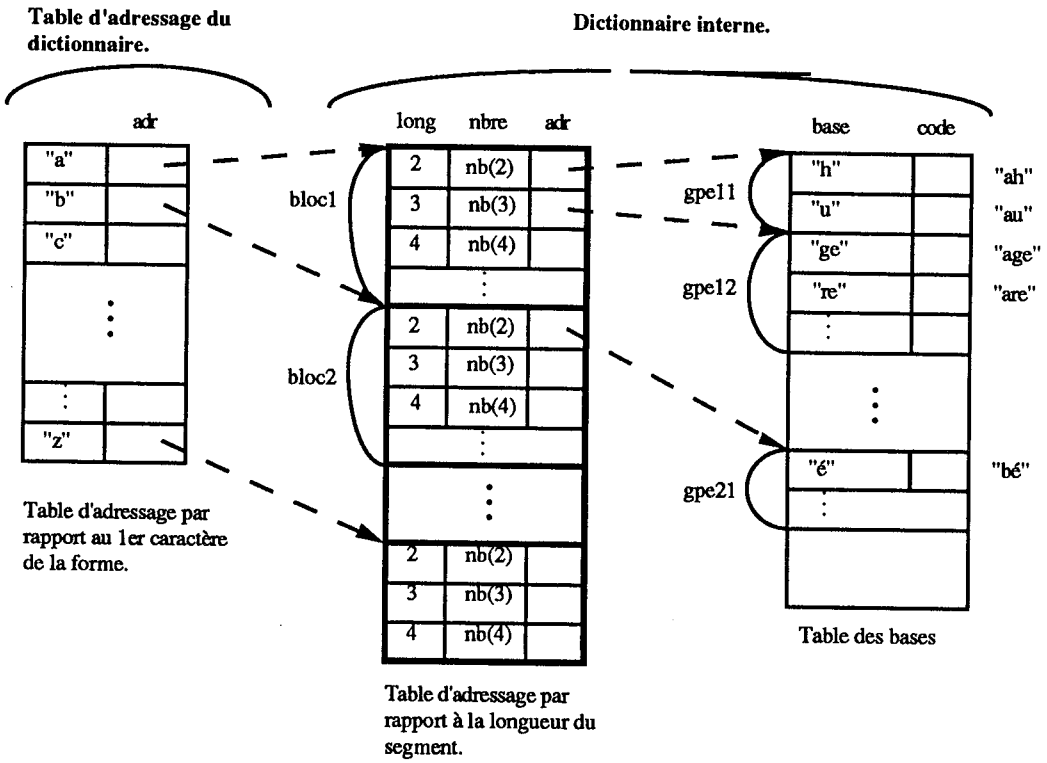
Nous pourrions trouver plus judicieux de vouloir gagner de la place en réduisant la taille du champs LETTRES (25 octets actuellement). Il semblerait raisonnable de conserver une taille de 7 à 8 octets pour la majorité des bases du français. Cependant, pour un petit nombre de cas particuliers (mots composés, tournures) cette taille est largement insuffisante. Il conviendrait donc d'étudier une nouvelle structure ou la taille des bases ne serait plus fixe.

Chaque enregistrement occupe donc 27 octets, ce qui donne 888300 octets pour les 32900 bases sans chaînage, 56700 octets pour les 2100 bases avec chaînage et 8400 octets pour les 2100 articles de la table d'index, soit au total 953400 octets. Cela représente un gain de 1041600 octets soit plus de 50% du volume initial.

Les insertions et suppressions d'éléments du dictionnaire intervenant dans un chaînage circulaire demandent une réorganisation de la table des index. Cela n'est pas un problème puisque sa faible taille (8,5 Ko) permet de la charger en mémoire centrale où sa mise à jour n'est pas coûteuse.

La nouvelle structure du dictionnaire PILAF a été implémentée sur IBM_PC et a permis de gagner une place considérable. Par contre, les avantages que cette nouvelle structure peut procurer en matière d'accélération du temps de traitement, au niveau de la génération des formes et de la lemmatisation (parcours des bases chaînées), n'ont pas encore été exploités sur cette machine.

4.1.3.2. Le dictionnaire ATEF



Les bases sont triées par ordre alphabétique croissant à l'intérieur de chaque groupe de longueur.

A un moment donné de l'analyse de la chaîne de caractères, un ensemble de caractères reste à analyser, soit "reste".

A un pas de la segmentation, l'analyse de "reste" se fait en trois temps:

- d'abord un hash-code parfait sur le premier caractère (de l'occurrence ou de la tournure à analyser),
- ensuite une itération sur les longueurs croissantes,
- une dichotomie sur l'ensemble des bases de la longueur cherchée.

Les trois tables étant chargées en mémoire centrale, la recherche d'une base est très rapide. A titre d'exemple, si l'on considère les 128 entrées de la table d'adressage, sachant que pour une entrée on a des longueurs allant de 2 à 20 au maximum, on défini ainsi 2432 classes dont on suppose qu'elles sont à peu près bien distribuées. Cela signifie que pour un dictionnaire de 50000 entrées (ce qui est déjà relativement important), on a en moyenne 20 éléments par classe, soit 20 entrées par groupe de longueur.

Calculons le nombre d'accès moyen nécessaire à la détermination de toutes les bases coïncidant avec la chaîne à analyser à partir de son premier caractère :

- 1 accès à la table d'adressage,
- 1 accès au bloc dont l'adresse est indiquée par la table d'adressage,

puis, il faut successivement accéder aux groupes de bases de longueur inférieure ou égale à la longueur de l'occurrence à analyser. En supposant qu'en moyenne une occurrence fait 12 caractères, il faut examiner 11 groupes (du groupe 2 au groupe 12). L'examen de chacun de ces groupes nécessite en moyenne :

- 1 accès au groupe de bases de la longueur voulue,
- environ 7 accès pour trouver la base (par dichotomie puis recherche séquentielle en fin de parcours) en supposant qu'un groupe comporte en moyenne 20 éléments.

Cela donne donc une moyenne de 1 + 1 + 11 (1 + 7) = 90 accès.

Cette structure privilégie la rapidité d'accès au détriment de la facilité de modification de la structure. Dans sa forme actuelle, les modifications (suppressions et insertions) dynamiques sont impossibles : il faut procéder aux modifications sur les dictionnaires source puis les recompiler.

Nous pourrions envisager d'exploiter cette structure, à condition de l'aménager en fonctions des contraintes du projet. En particulier, il n'est pas possible de charger la structure complète en mémoire centrale du Macintosh ; on en chargerait seulement une partie, à savoir :

- la table d'adressage par rapport au premier caractère (< 1 K),
- la table d'adressage par rapport à la longueur (< 10 K).

Si on voulait permettre les modifications dynamiques,

- on ajouterait à chaque enregistrement de la table d'adressage par longueur un pointeur sur l'enregistrement longueur suivant pour pouvoir créer un nouveau groupe de longueur facilement,
- on laisserait des emplacements vides en fin de chaque groupe de bases dans la table des bases, pour pouvoir ajouter de nouvelles bases.

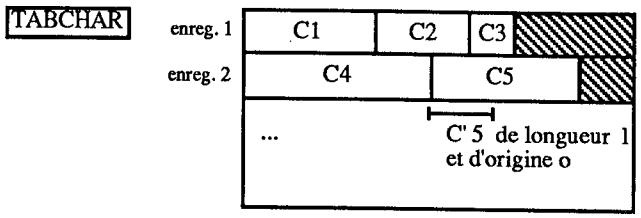
Mais cette structure ainsi aménagée perdrait un peu de son efficacité en matière de recherche, puisque la majorité des accès en mémoire centrale deviennent alors des accès disque.

4.2. Solution proposée

L'étude d'une nouvelle structure du dictionnaire PILAF a essentiellement été motivée par le souci d'un gain de place. Mais il ne s'agissait pas de gagner de la place en figeant le dictionnaire ou en occultant les considérations de performance en terme de temps d'accès à ses éléments.

Notre idée a donc été de stocker les bases du dictionnaire dans un tableau de caractères. Les bases y sont factorisées selon les chaînes de caractères communes. Ce tableau pourra être constitué comme suit :

- l'ensemble des bases a préalablement été trié dans un ordre décroissant,
- chaque base est stockée dans le tableau de caractères TABCHAR à partir du premier espace libre si elle n'est pas incluse dans la dernière base stockée, sinon elle n'est pas stockée.



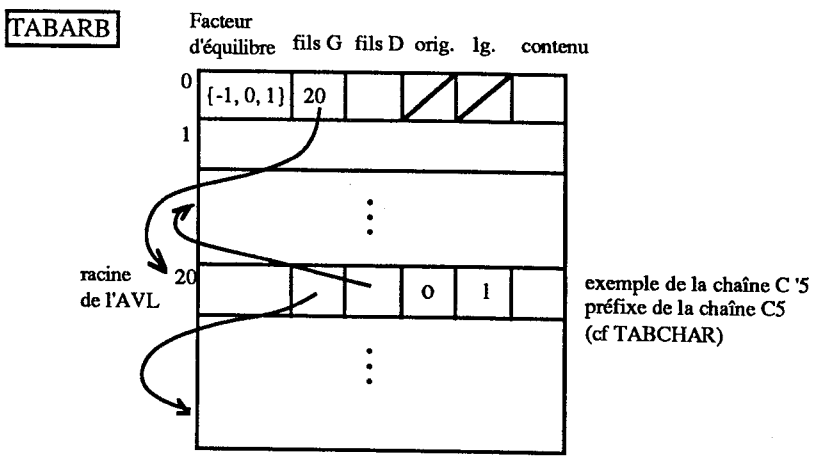
On laisse des "trous" quand la base à stocker demande plus de place qu'il n'en reste sur l'enregistrement courant.

Chaque base est identifiée par une clé. On range la référence à cette clé dans un arbre binaire TABARB. Nous avons choisi un arbre binaire équilibré, autrement appelé arbre AVL [KNUT73].

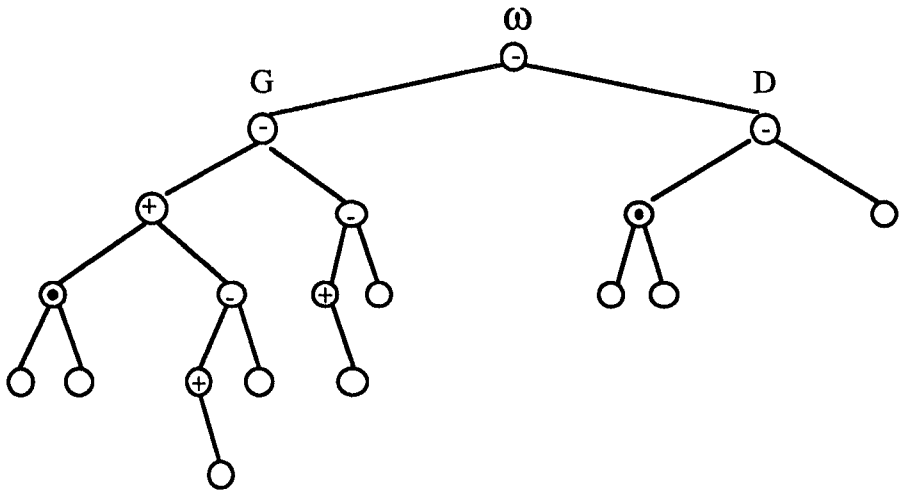
On appelle hauteur d'un arbre la longueur du plus long chemin de la racine de l'arbre à une feuille. soit n un nœud de l'arbre A. Nous notons $H_n(G)$ la hauteur du sous-arbre gauche de n, $H_n(D)$ la hauteur du sous-arbre droit de n.

Un arbre binaire A est dit équilibré si, pour tout nœud n de A, $| H_n(G) - H_n(D) | \leq 1$.

En fait, dans l'AVL, chaque clé est identifiée par le couple (Origine, Longueur) de la chaîne de caractères de la base dans TABCHAR.



Cette table représente en fait un arbre AVL, qui peut être le suivant :



L'enregistrement 0 de TABARB représente la racine ω de l'arbre binaire qui a pour fils gauche un arbre AVL de racine G et qui a pour fils droit un arbre AVL de racine D.

- L'arbre de racine G représente les clés des chaînes présentes dans TABCHAR, triées par ordre croissant des chaînes lorsqu'on parcourt l'arbre de manière infixée.
- L'arbre de racine D représente la liste des "trous" dans TABCHAR, triée par longueur, origine croissantes. Ces "trous" viennent des suppressions.

Examinons plusieurs opérations dans un dictionnaire organisé selon cette structure.

Recherche d'une base

La recherche d'une base b est réalisée par un parcours du sous-arbre gauche de la racine ω de TABARB.Pour chaque nœud rencontré, on accède à la chaine de caractère C de TABCHAR identifiée par le couple (Origine, Longueur) du nœud. On compare la chaine C avec la base b recherchée. Si elles sont égales, la recherche est fructueuse. Si elles sont différentes, deux cas peuvent se produire. Si le nœud courant de l'arbre est une feuille, alors la recherche est finie et la base B n'existe pas dans le dictionnaire. Sinon, il faut "descendre" dans l'arbre. Si C > b, alors on accède au fils gauche du nœud courant qui devient le nouveau nœud courant. Si C < b, alors on accède au fils droit du nœud courant qui devient le nouveau nœud courant. Puis, on recommence le même processus pour le nouveau nœud courant.

Suppression d'une base

La suppression d'une base b de longueur L_b s'effectue en trois phases.

La première phase consiste à rechercher la base b dans le dictionnaire selon la méthode décrite ci-dessus. Au cours de cette recherche, on identifie le nœud représentant le plus grand préfixe de la base b à supprimer, soit p. Ce préfixe p a pour origine O et pour longueur L_p. On identifie la base q dont b est préfixe si une telle base existe.

Une deuxième phase consiste à supprimer la clé identifiant la base b dans l'arbre, puis à marquer (par exemple par des zéros) les caractères à supprimer dans TABCHAR si la base b n'est préfixe d'aucune base q. Il s'agit des caractères de la base b n'appartenant pas p, et plus précisément des caractères situées à partir de la position $O + L_p$ jusqu'à la position $O + L_b - 1$. Les caractères supprimés engendrent ce que nous avons appelé un trou dans TABCHAR.

La troisième phase n'a lieu que si des caractères ont été supprimés dans TABCHAR. Elle consiste à insérer un nœud (Origine, Longueur) identifiant le trou dans le sous-arbre droit de la racine ω de TABARB. Il ne faut pas oublier de fusionner deux "trous" contigus. Cela se produit lorsqu'une suppression fait apparaître un trou à côté d'un trou déjà existant.

Insertion d'une base

L'insertion d'une base b de longueur L_b s'effectue en plusieurs phases.

On recherche d'abord la base b dans l'arbre AVL des chaînes selon la méthode décrite plus haut. Si elle est trouvée, il n'y a pas d'insertion à faire.

Sinon, on recherche dans l'arbre des "trous" s'il existe dans TABCHAR un trou de longueur L_b. S'il existe un tel "trou", on y insère la base b. On supprime le trou de l'arbre des trous (on rééquilibre l'arbre si nécessaire). On met à jour l'information origine de la clé de la base b dans l'arbre des chaînes. Si un tel trou n'existe pas, on ajoute la base b en fin du tableau TABCHAR. On met à jour l'information origine de la clé de la base b dans l'arbre des chaînes. On détermine la position, dans l'arbre, de la clé de la nouvelle base et on l'insère. Puis, on équilibre l'arbre si nécessaire.

Réorganisation de la structure

Pour maintenir l'optimisation du compactage du tableau de caractères, une réorganisation de la structure s'impose lorsque l'on a ajouté en fin du tableau TABCHAR beaucoup de chaînes dont des chaînes existantes sont préfixes (la fréquence des réorganisations serait à évaluer par calcul).

Pour réorganiser la structure, il faut parcourir l'arbre AVL des chaînes en infixé droit (donc en ordre décroissant des chaînes) et recréer un nouveau tableau de caractères TABCHAR, en mettant à jour les informations (Origine, Longueur) sur chaque nœud. On remplace l'arbre des trous par un arbre vide.

Cette solution serait à approfondir. Notamment, il faudrait voir si l'algorithme de l'automate PILAF pourrait s'adapter aisément à cette structure et comment cette structure pourrait être gérée en mémoire secondaire.

On pourrait également étudier pour PILAF la solution consistant à organiser les clés en B-tree, donc à utiliser un arbre n-aire plutôt qu'un arbre binaire équilibré, et comparer les deux solutions.

Chapitre III
Réalisation

1. Le contexte et l'apprentissage

1.1. Logiciels

Au départ du projet, nous constatons la situation suivante:

- le lemmatiseur PILAF réalisé par les participants au "projet MacPilaf", à partir de l'analyseur développé par l'équipe TRILAN, tourne sur compatible PC en TURBO Pascal [CHAP89b],
- l'outil d'accès à des bases documentaires WinTool™ est un accessoire de bureau sur Macintosh écrit en TML Pascal.

ODILE, nous l'avons vu plus haut, doit être un accessoire de bureau, et nous avons choisi de l'écrire en THINK Pascal™.

Nous avons donc dû porter le lemmatiseur PILAF sur Macintosh en THINK Pascal™ 3.01 et WinTool™ vers THINK Pascal™ 3.01. Le portage de PILAF a été réalisé par mes soins avec l'aide de l'équipe TRILAN, celui de WinTool™ a été assuré par la société WinSoft, concepteur du produit.

1.2. Système

Pour les développeurs, la tâche à réaliser est moins évidente sur un Macintosh que sur une tout autre micro-ordinateur.

Le Macintosh est en effet connu pour être une machine difficile à programmer, en raison surtout de la sophistication de son interface utilisateur, des routines nombreuses et complexes que cette machine propose au programmeur. Mais cela en vaut la peine puisqu'ainsi la plupart des logiciels sur Macintosh suivent un modèle d'interface standard, si bien que tout utilisateur peut apprendre à utiliser une nouvelle application presque immédiatement.

La gestion de la mémoire du Macintosh est particulière également, et constitue une difficulté supplémentaire pour le programmeur.

1.2.1. Interface utilisateur et programmation par événements

L'interface utilisateur graphique et les logiciels pilotés par événements sont deux concepts fondamentaux du Macintosh.

1.2.1.1. Interface utilisateur

Sur Macintosh, plus n'est besoin de connaître la syntaxe particulière des commandes du système d'exploitation et des applications. Les commandes possibles à un instant donné sont visibles à l'écran sous forme de choix dans des menus, de boutons, de cases à cocher, etc. Cette présentation peut s'enrichir d'icônes.

L'interface vise à reproduire le monde réel dans lequel l'utilisateur peut désigner et déplacer des objets, utiliser des accessoires tels qu'une calculette, une horloge, des ciseaux. Sur le Macintosh, les actions de l'utilisateur s'effectuent en général avec la souris qui sert de substitut électronique de la main.

Cette interface exploite à fond l'affichage graphique qui a été choisi sur le Macintosh. Ce choix a été guidé par l'idée qu'un graphique ou un dessin véhiculent souvent plus d'information que du texte et par le fait que ce type d'affichage propose une plus grande souplesse dans la présentation des textes en terme de taille, de style, de police de caractères.

L'affichage graphique est plus complexe que l'affichage de texte uniquement. Les systèmes graphiques, contrairement aux systèmes à affichage alphanumérique, doivent gérer tous les objets destinés à apparaître à l'écran, y compris les caractères, sous forme de matrice de points. Le système d'exploitation du Macintosh propose un ensemble de fonctions et procédures standard réparties dans plusieurs "managers" parmi lesquels QuickDraw, spécialisé dans l'affichage graphique.

1.2.1.2. Logiciels pilotés par événement

Le système d'exploitation du Macintosh gère une file d'événements pour la ou les applications ouvertes. Toute application contient une boucle principale de lecture des événements de la file.

Cette boucle contient des instructions de traitement des événements en fonction de leur

nature (bouton de la souris appuyée, sélection dans un menu, action dans une fenêtre, etc). Ces instructions sont le plus souvent des appels à des fonctions et procédures internes.

Examinons le schéma de programme suivant:

```
begin
        Initialize;
        repeat
          SystemTask;
          if GetNextEvent(eventMask, theEvent)  then
               HandleEvent(theEvent);
        Until Done
        CleanUp;
end.
```

C'est la structure de base de la plupart des applications Macintosh.

Après l'initialisation de l'environnement, le programme entre dans une boucle qui prend un nouvel événement dans la file d'événements (procédure GetNextEvent de la ToolBox) et le traite (HandleEvent). Cela se reproduit jusqu'à la survenue d'un événement particulier (par exemple sélection de la commande Quitter dans un menu) qui bascule à "vrai" une certaine condition (Done).

La programmation pilotée par événements permet de construire des applications sans mode ; cela signifie que toutes les commandes sont généralement disponibles à tout instant. Il faut donc prévoir de les traiter. En particulier, le programmeur doit traiter tous les événements pouvant survenir, dans la procédure "HandleEvent".

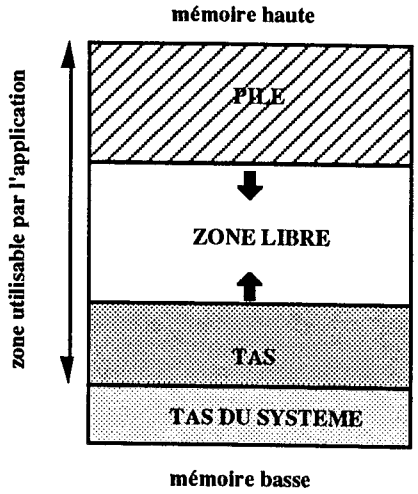
```
procedure  HandleEvent ( theEvent : EventRecord);
   begin
      case  theEvent.What  of
        mouseDown    :  DoMouseDown (theEvent);     { bouton souris enfoncé)
        KeyDown      :  DoKeyPress (theEvent);      { touche clavier enfoncée}
        autoKey      :  DoKeyPress (theEvent);      { touche maintenue enfoncée}
        updateEvt    :  DoUpdate (theEvent);        { fenêtre à mettre à jour}
        activateEvt  :  DoActivate (theEvent);      { fenêtre à activer/désactiver}
      end
   end;
```

1.2.2. Gestion de la mémoire

Sur Macintosh, comme sur tout ordinateur, la mémoire est divisée en un certain nombre de zones. On peut distinguer le tas (heap) et la pile (stack).

Les blocs mémoire de la pile sont alloués et libérés selon l'ordre LIFO (Last In First Out), tandis que ceux du tas peuvent l'être dans n'importe quel ordre selon les besoins de l'application.

Le schéma qui suit illustre la localisation du tas et de la pile dans la mémoire du Macintosh [INSM87].

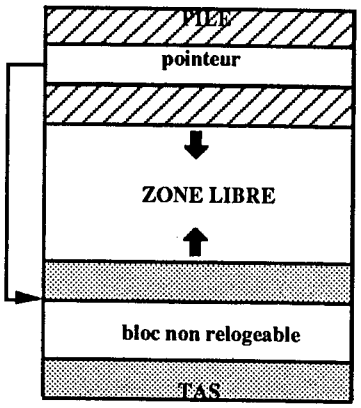


Le "Memory Manager", qui est une partie du système d'exploitation du Macintosh, est chargé de la gestion des tas, tas du système et tas de l'application.

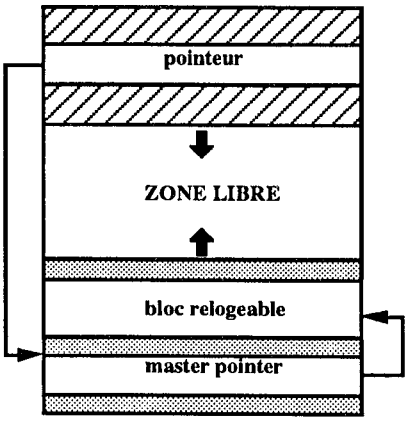
L'espace dans le tas est divisé en blocs contigus. Ils peuvent être alloués ou libres.

Un bloc alloué peut être relogeable ou non relogeable ; un bloc relogeable peut être déplacé dans le tas afin de libérer de la place pour d'autres blocs. Un bloc non relogeable ne peut pas être déplacé.

Un bloc non relogeable est référencé par un pointeur.



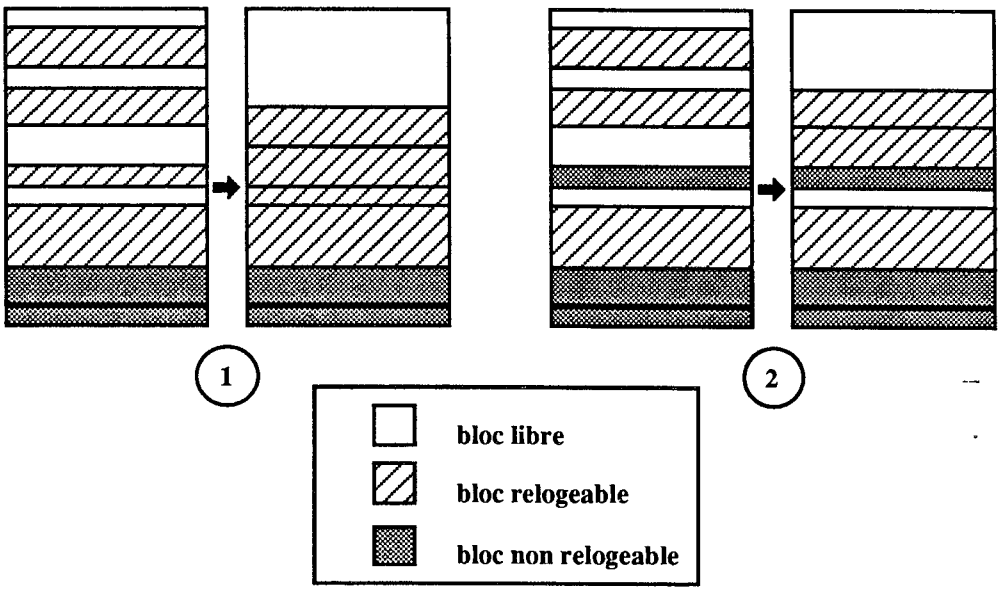
Un bloc relogeable est référencé par un pointeur sur un pointeur que nous appellerons poignée dans la suite de cet exposé ("handle" en terme Macintosh).



Le pointeur maître ("master pointer") ne bouge pas.

S'il a besoin de place, le gestionnaire de mémoire ("Memory Manager") peut déplacer le bloc relogeable. Il met alors à jour le pointeur maître avec la nouvelle adresse de celui-ci.

Les schémas suivants illustrent le comportement des deux types de bloc lorsque le gestionnaire de la mémoire entreprend un compactage en vue de libérer de l'espace mémoire.



Dans la cas 1, même si des blocs non relogeables ont été alloués, ils l'ont été dans la mémoire basse, ce qui ne gêne pas le compactage. On récupère toute la mémoire libre en un ensemble de blocs contigus.

Dans le cas 2 par contre, les blocs non relogeables ont également été alloués dans le milieu du tas, ce qui a pour conséquence une fragmentation de la mémoire : l'espace libre ne pourra

jamais être récupéré en un seul bloc. l'allocation de futurs blocs risque de devenir impossible. C'est pourquoi, il est fortement conseillé d'allouer dans le tas des blocs relogeables plutôt que non relogeables.

Le gestionnaire de mémoire propose pour cela un ensemble de procédures et fonctions pour créer et manipuler les poignées. Cependant, une poignée est un type de donnée à manipuler avec précaution, étant donné que le gestionnaire de la mémoire peut déplacer la structure à laquelle elle est rattachée selon ses besoins. Plus précisément, il ne faut jamais considérer l'adresse d'une structure référencée par une poignée valable à un instant donné comme éternellement valide.

Nous détaillerons les précautions à prendre au paragraphe 2.1.1.1.2 page 111, qui décrit les modifications apportées à PILAF pour l'adapter au Macintosh.

1.3. Méthodologie retenue

Notre objectif a été de réaliser un prototypage d'ODILE sur Macintosh, et plus précisément un prototypage horizontal dans la mesure où toutes les fonctionnalités d'ODILE sont ébauchées (contrairement au prototypage vertical où l'on réalise complètement les quelques fonctions représentatives du logiciel). Ce choix correspond à une volonté de voir comment réagit l'utilisateur face à un tel logiciel. De plus, le prototypage horizontal a l'intérêt de mener à la découverte de la structure interne des logiciels variés que notre outil intègre.

La réalisation s'est faite en plusieurs étapes que nous détaillons dans les paragraphes qui suivent.

2. Contenu de la première version

Cette partie décrit ce qui a été réalisé concernant l'adaptation des logiciels intégrés et le développement des fonctions propres et de l'interface utilisateur de la première version d'ODILE, ainsi que l'architecture logicielle mise en place pour notre outil.

2.1. Adaptation des logiciels

Une première tâche a consisté à porter PILAF sur Macintosh à l'état brut, c'est-à-dire sans les aménagements particuliers que nécessite son intégration dans ODILE. En tout état de cause, il est intéressant de disposer de cet outil sur Macintosh, en vue d'autres applications. Puisque le

langage de programmation était choisi à cette étape du projet, le lemmatiseur a simplement été converti de Turbo Pascal vers THINK Pascal™. Cette première tâche a été l'occasion d'expérimenter les particularités du Macintosh, de nous familiariser avec les concepts propres à cette machine.

Une seconde tâche a ensuite été de d'effectuer sur le code de PILAF les aménagements nécessaires à son intégration dans ODILE.

De façon symétrique, il a fallu intervenir, de façon moindre, sur le code de WinTool™. Il est à noter que nous nous sommes permis de faire quelques aménagements dans WinTool™, dans la mesure où, pour des raisons techniques, son code nécessitait des modifications en vue de son intégration dans ODILE.

2.1.1. Adaptation de PILAF

2.1.1.1. Version brute

Nous avons procédé en deux étapes. Un premier travail a consisté à adapter la syntaxe des programmes à celle de THINK Pascal™, de façon à obtenir des fichiers compilables. A lui seul, ce travail a demandé près de deux semaines. L'adaptation des programmes à la gestion de la mémoire particulière sur Macintosh a ensuite demandé un travail encore plus conséquent.

2.1.1.1.1. Adaptation à THINK Pascal™

Nous avons constaté de nombreuses différences entre Turbo Pascal [TURB86] et THINK Pascal™ [THIN90]. Ces différences portent notamment sur :

- les ouvertures de fichier,

exemple :

Turbo Pascal	THINK Pascal™
assign (Var. fichier, Nom fichier);	**open (Var. fichier,Nomfichier);**
open (Var. fichier);	

- les opérations autorisées sur des opérandes,

. Opérande de type String[...] + Opérande de type 'Chaîne de car.'

autorisé en Turbo Pascal, refusé en THINK Pascal™

. OR, AND

autorisé sur booléens et masques de bits en Turbo Pascal

refusé sur masques de bits en THINK Pascal™ (utiliser BitOr, BitAnd, ...)

- les opérations sur les bits,

  - **shl, shr**, ... en Turbo Pascal,
  - **BSL, BSR**, ... en THINK Pascal™

- les tests de valeurs de caractère,

  exemple :

  if c = # 13 then ...

  autorisé en Turbo Pascal

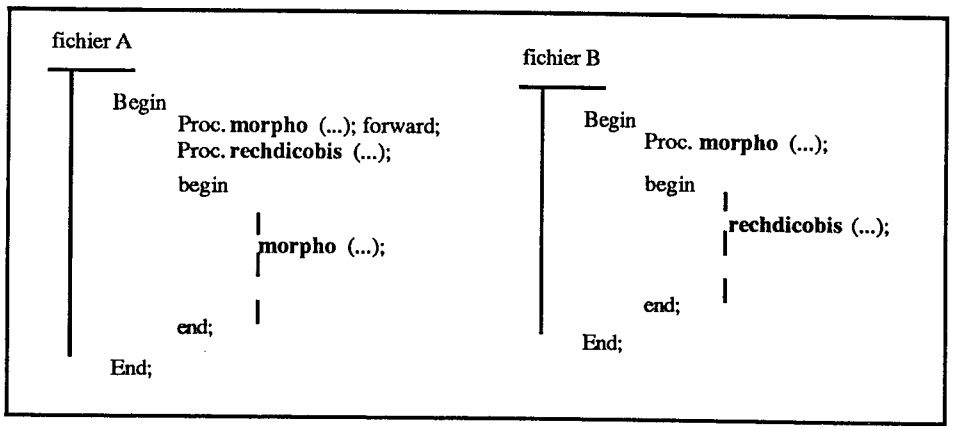
  refusé en THINK Pascal™

Mais la différence fondamentale est que THINK Pascal™ n'accepte pas de fichier inclus dans un programme : toute procédure ou fonction appelée par un programme, ou une autre procédure ou fonction, doit appartenir à une unité (Unit). Une "Unit" est un fichier devant respecter une syntaxe particulière.

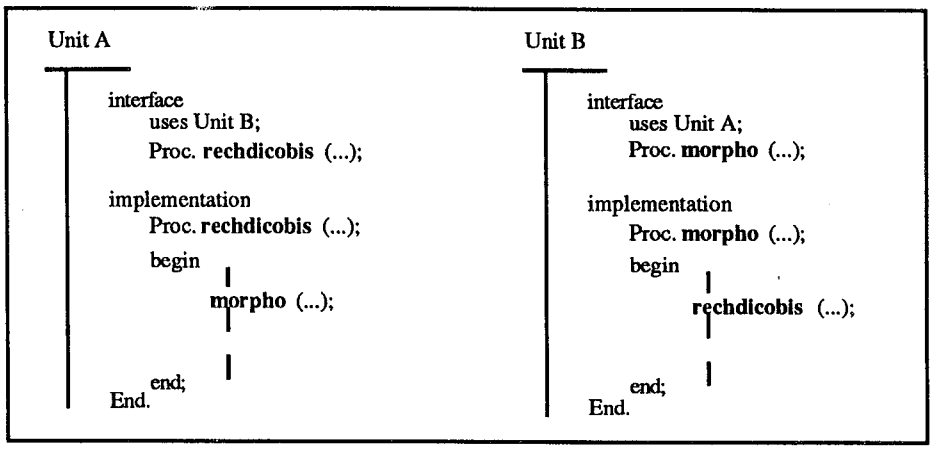
Or, les programmes sources de PILAF en Turbo Pascal n'étaient pas construits par assemblage d'unités. Il a donc fallu reprendre chacun des fichiers de procédures et de fonctions précédemment inclus dans le module PILAF en Turbo Pascal pour les transformer en "Units".

Il a également fallu revoir la découpe logique de l'ensemble des procédures et fonctions en fichiers. La raison en est simple : toute procédure ou fonction utilisée dans une unité doit avoir été déclarée auparavant.

Examinons l'exemple des deux fichiers suivants :

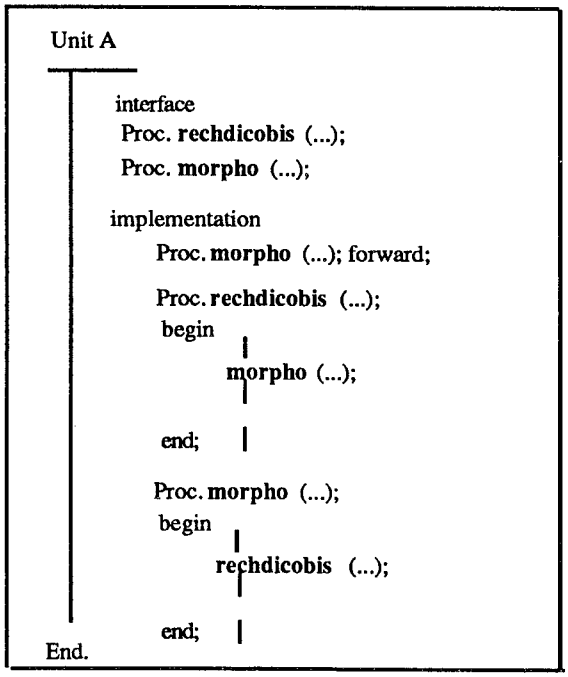


Cette configuration, que l'on peut trouver dans la version Turbo Pascal de PILAF sur PC, n'est pas transposable directement en unités THINK Pascal™.



Le schéma de programme ci-dessus n'est pas valable.

Dans le cas d'utilisation mutuelle (Unit A utilise Unit B, et Unit B utilise Unit A), on est obligé de n'avoir qu'une seule unité, c'est-à-dire qu'il faut refondre le fichier A et le fichier B originels en une seule unité.



2.1.1.1.2. Adaptation au Macintosh

A l'origine, sur micro-ordinateur compatible IBM-PC, le module lemmatiseur de PILAF comportait de nombreuses zones de mémoire référencées par des pointeurs. La raison en est que, sur compatible IBM-PC de même que sur Macintosh, la taille du segment de données d'une application est limitée et que PILAF manipule un très grand nombre de données, essentiellement sous forme de tableaux.

Il fallait donc, pour ces données, allouer des blocs dans un espace beaucoup moins

restreint, le tas. Sur PC, la seule méthode était d'allouer des pointeurs sur ces données.

Mais sur le Macintosh, lorsqu'on alloue des pointeurs sur des données, on court, dans certains contextes, un risque de fragmentation de la mémoire et d'impossibilité d'allocation de blocs durant l'exécution du programme.

Dans le cadre de notre projet par exemple, les procédures du lemmatiseur extrait de PILAF sont destinées à être exploitées dans un accessoire de bureau. Or, nous verrons un peu plus loin qu'un accessoire de bureau est toujours hébergé par une application hôte. Cela signifie, entre autres, que les structures de données dynamiques qu'il utilise vont s'ajouter aux structures de données de l'application hôte dans le tas de l'application. Si les structures de données de l'accessoire ne sont pas relogeables, elles risquent de perturber le fonctionnement de l'application.

En conséquence, toutes les structures référencées à l'origine par des pointeurs, ont été modifiées pour être référencées par des poignées ("handles").

La mise en place des poignées peut être réalisé de 2 manières :
- soit on crée de nouveaux types (ex. de gauche),
- soit on effectue des coercions de type "à la demande" (ex. de droite).

```
Type
    elem =Record
            field : ...;
        end;
    TTab = Array [1..Max] of elem;
    TabPtr =^TTab;          *** 1 ***
    TabHdl = ^TabPtr;       *** 1 ***
Var
    Htab : TabHdl;
    Tab : TTab;
begin

    Handle (Htab) := NewHandle (SizeOf (Tab));   *** 2 ***
    if HTab^^[i] . field ...
end.
```

```
Type
    elem =Record
            field : ...;
        end;
    TTab = Array [1..Max] of elem;
Var
    Htab : Handle;
    Tab : TTab;
begin

    Htab := NewHandle (SizeOf (Tab));
    if TTab (HTab)^^[i] . field ...      *** 3 ***
end.
```

*** 1 *** *Nouveaux types*
*** 2 *** *Coercion à la création*
*** 3 *** *Coercions à la manipulation*

On constate sur cet exemple que la création d'une poignée est réalisée grâce à une primitive particulière, NewHandle(...). On constate également que la manipulation des poignées nécessite des détypages :

- un seul pour transformer la variable de type pointeur de pointeur TabHdl en variable de type Handle (schéma de gauche pour la création de la poignée Htab),
- plusieurs pour transformer la variable de type Handle en variable de type TTab (schéma de droite pour la manipulation des éléments du tableau référencés par la poignée HTab).

Il s'en est suivi de nombreuses modifications du code du module de lemmatisation, d'autant plus nombreuses que la manipulation de poignées nécessite certaines précautions. En effet, on rencontre parfois des problèmes d'adressage. En voici quelques exemples.

- myHandle^^.code := myFunction (x);
Avec certains compilateurs, l'adresse valeur de myHandle^^ est stockée. Puis myFunction (x) est activée. Il risque d'y avoir une affectation à une adresse ne correspondant plus à l'adresse originelle de myHandle^^.code si la fonction contient des instructions qui provoquent une réorganisation de la mémoire.

- With myHandle^^ do ...
Une copie du pointeur maître est faite. On risque également d'avoir des erreurs d'adressage si certaines instructions subordonnées à l'instruction "With" modifient les adresses en mémoire.

- TrierTableau (myHandle^^. ElemTab);
C'est l'adresse de l'élément de tableau qui est empilée. On aura un problème d'adressage par la suite si la procédure de tri appelle une routine d'allocation mémoire qui réorganise la mémoire : on mémorisera un résultat à une adresse qui risque de n'être plus la bonne.

Pour éviter ces déboires, la solution est de verrouiller les blocs du tas référencés par des poignées lorsque la manipulation de ceux-ci comporte un risque.
Pour ce faire, le gestionnaire de mémoire propose deux primitives.
- HLock (myHandle) verrouille la structure référencée par myHandle, c'est-à-dire qu'elle empêche son déplacement dans la mémoire,
- HUnLock (myHandle) déverrouille la même structure.

Cependant, il ne faut pas faire ces "verrouillages" et "déverrouillages" à tort et à travers. Notamment dans le cas d'appels de procédures imbriqués qui manipulent la même poignée, on court deux risques :
- celui de verrouiller toute la mémoire si toutes les poignées restent constamment bloquées (appels récursifs),
- celui d'avoir des erreurs d'adressage en déverrouillant trop tôt dans une procédure appelée une poignée manipulée également par la procédure appelante.

Pour éviter cela, il faut tester l'état (verrouillé ou non verrouillé) de la poignée et sauvegarder cet état (HGetState(myHandle)). On bloque la poignée si son état est non verrouillé. Puis on exécute les instructions sur cette poignée. En fin de procédure, on restaure l'état initial de la poignée (HSetState(myHandle)). Le manque d'informations précises sur le fonctionnement du compilateur de THINK Pascal™, nous à conduit, dans le doute, à prendre toutes les précautions.

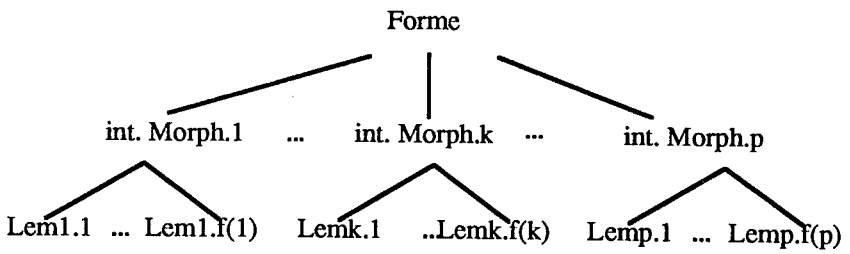
La mise en œuvre de tout cela a entraîné des modifications conséquentes du code du module de lemmatisation de PILAF. Nous y avons en effet consacré environ 1 mois.

Neuf tables de données référencées par des pointeurs ont été modifiées pour être référencées par des poignées : il s'agit des tables destinées à mémoriser les données nécessaires au processus de lemmatisation (règles, masques de règles, modèles, variables, masques de variables, catégories lexicales, contraintes de lemmatisation, fichier des terminaisons, fichier maître des bases).

La gestion de ces tables est peu changée. Les procédures qui les manipulent ne sont pas fondamentalement différentes. Seules quelques instructions garantissant une bonne gestion mémoire en présence des poignées ont été insérées.

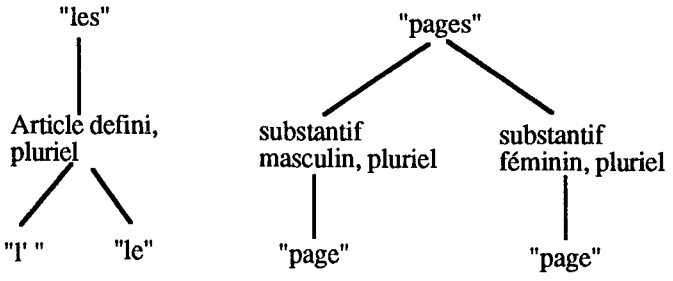
Par contre, la structure de sortie du lemmatiseur a radicalement changé. Cette structure permet de fournir à l'utilisateur l'ensemble des lemmes générés pour chacune des interprétations morphologiques données pour chaque forme analysée. On appelle interprétation morphologique d'une forme l'ensemble {classe lexicale, variables morphologiques} déterminé par l'analyse pour cette forme.

La figure qui suit illustre cette structure.

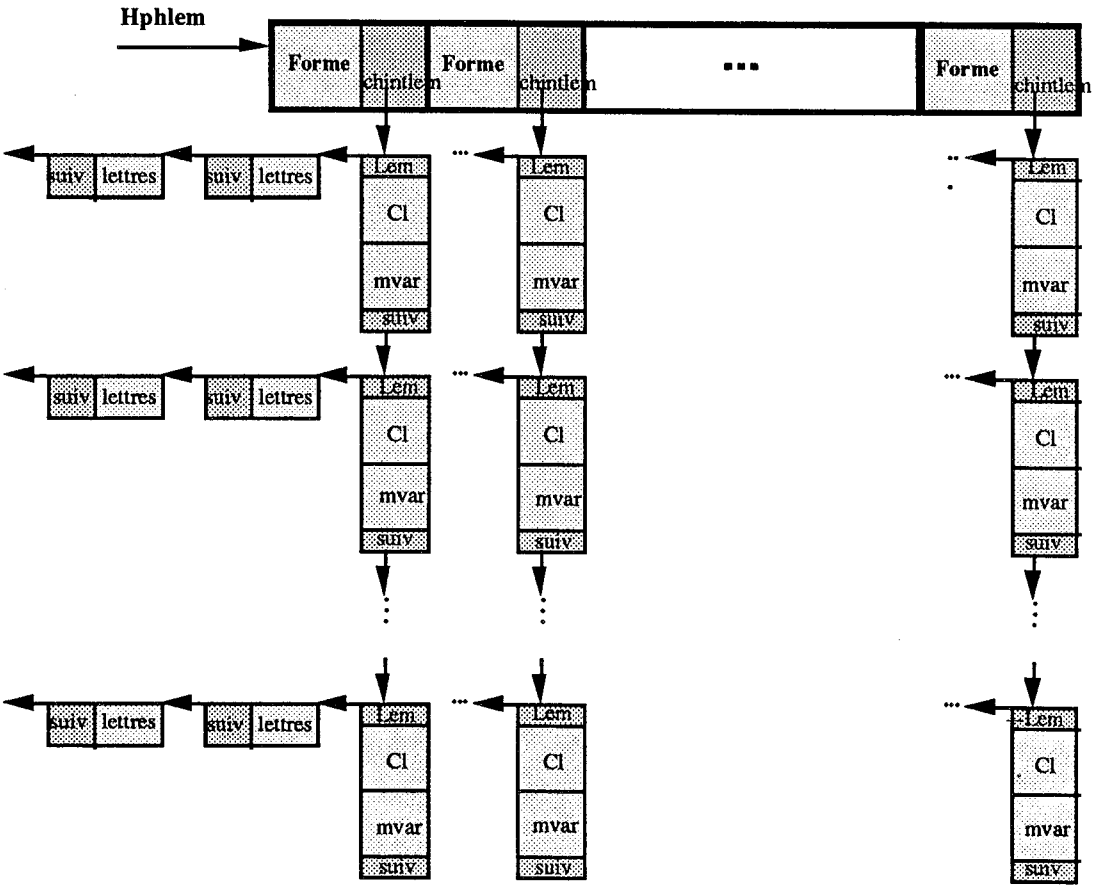


Cette structure prévoit plusieurs lemmes pour un même interprétation morphologique. Mais, sauf de très rares exceptions ("les", article défini, donne les 2 lemmes "l'" et "le"), il n'y en a en général qu'un seul.

Exemple: structure interne du résultat de l'analyse morphologique de la chaîne "les pages".



Le lemmatiseur issu du projet MacPILAF prévoyait l'analyse d'un maximum de 30 occurrences(ou formes). La structure interne de son résultat était donc un tableau fixe de 30 postes, référencé par le pointeur `Hphlem`. Cette structure est décrite par le schéma suivant :



Notons ici que, dans la version du lemmatiseur PILAF reprise, certains des identificateurs utilisés ne sont pas très bien choisis et peuvent prêter à confusion.

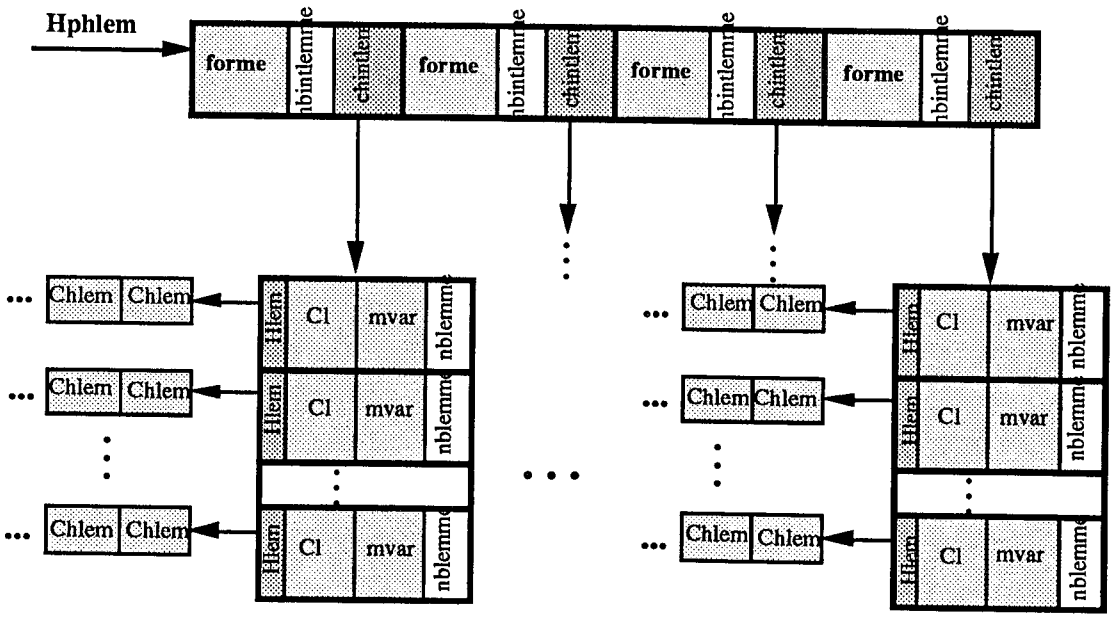
Nous précisons donc que :

- le champ "Lem" désigne un pointeur sur la liste des lemmes correspondant à une interprétation morphologique,
- le champ "lettres" représente la chaîne des caractères composant un lemme.

Cette structure était bien adaptée au stockage des résultats de lemmatisation : on ne connait pas a priori le nombre d'interprétations morphologiques pour chaque occurrence ("forme") et pas plus le nombre de lemmes possibles pour chaque interprétation morphologique de l'occurrence ; la qualité dynamique des listes linéaires convient donc bien.

Pour nous, cette structure n'est pas exploitable sur Macintosh. Une liste linéaire est en effet constituée d'un ensemble de pointeurs référençant chacun un bloc de mémoire et chaînés les uns aux autres. Il s'agit donc d'une structure non relogeable.

Il a donc été nécessaire de modifier cette structure de donnée pour qu'elle soit relogeable. Chaque liste linéaire a donc été remplacé par un tableau dynamique référencé par un "handle". Cela donne la nouvelle structure suivante, référencée par la poignée `Hphlem` :



Il est à noter que maintenant le tableau référencé "`Hphlem`" est un tableau dynamique, c'est-à-dire que le nombre d'occurrences traitables par le lemmatiseur n'est plus limité.

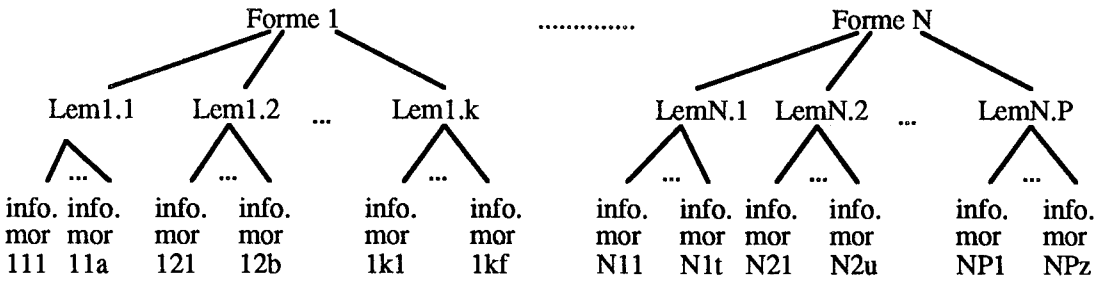
`Chintlem` et `Hlem` sont des "handles" référençant des tableaux dynamiques. `Chintlem` référence un tableau des interprétations morphologiques de la forme concernée ; `Hlem` référence un tableau de lemmes `Chlem` correspondant à chaque interprétation morphologique.

`nbintlemme` et `nblemme` représentent respectivement le nombre d'éléments du tableau référencé par `Chintlem` et le nombre d'éléments du tableau référencé par `Hlem` : ces valeurs sont utiles pour le parcours des tableaux.

Bien entendu, nous avons dû modifier considérablement le code des unités du

lemmatiseur plus particulièrement chargées de la génération des lemmes (unité `GenerLem` du fichier `GenerLem.lem`), puisque la gestion de la nouvelle structure devient plus complexe que celle des listes linéaires.

En fin d'analyse, cette structure de sortie est lue pour créer un fichier texte (`Fmotltex`) où les résultats sont factorisés selon les lemmes communs à chaque interprétation morphologique. Le schéma suivant illustre cette factorisation.



Le fichier texte des lemmes factorisés est ensuite affiché à l'écran . C'est donc sous cette forme factorisée que les résultats sont présentés à l'utilisateur. La procédure effectuant cette factorisation (procédure `Ecrfic` de l'unité `GenerLem`) a également dû être modifiée pour prendre en compte la structure interne des résultats de lemmatisation adaptée au Macintosh.

2.1.1.1.3. Portage des fichiers de données

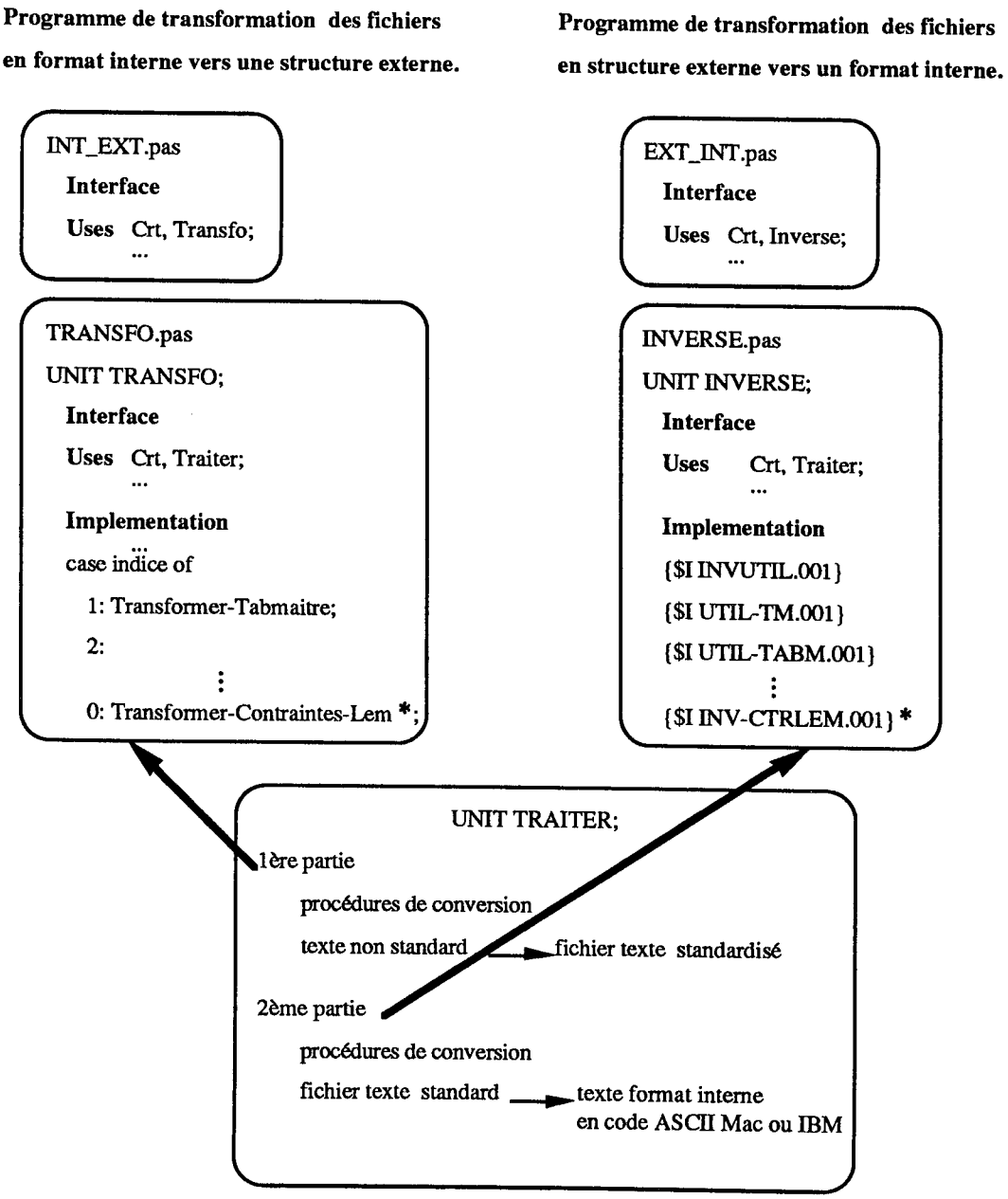
Pour effectuer le portage du module de génération de PILAF de l'IBM PC vers le Macintosh, les participants au projet MacPILAF [CHAP89b] avaient défini une forme standard pour les programmes sources et les fichiers de données utilisés par ces programmes. Cette forme standard est appelée structure externe des données.

En effet, la différence des codes ASCII entre les deux systèmes d'exploitation, ainsi que la différence induite de l'ordre lexicographique, impose une conversion préalable des fichiers à transférer de la forme binaire interne vers une forme externe commune aux deux systèmes.

Le programme INT_EXT.pas réalise cette tâche ; il est paramètré par un booléen indiquant si l'on part d'une structure interne IBM ou Macintosh. Ce programme permet en outre de fournir, pour les fichiers de données, une présentation lisible par l'utilisateur de chaque enregistrement. Le transfert du fichier converti peut alors avoir lieu. Puis, il faut rétablir la structure interne du fichier sur la nouvelle machine. Un autre programme, INT_EXT.pas, réalise cela ; de même, un booléen indique si l'on souhaite obtenir une forme interne sur Macintosh ou sur IBM.

Au total, ces programmes permettent le transfert de programmes sources et de fichiers de données dans les deux sens (Mac -> IBM, IBM -> Mac). Ils permettent aussi de visualiser le contenu des fichiers de données sous forme d'un fichier texte.

Mais le lemmatiseur mis au point sur IBM-PC n'ayant, à ce moment là, pas été porté sur Macintosh, les étudiants n'ont pas ressenti le besoin de transférer sur Macintosh le fichier des contraintes de lemmatisation utilisé seulement par ce module. Sa mise sous forme standard n'a pas non plus été prévue dans les programmes. Il a donc fallu adjoindre une procédure de traitement du fichier des contraintes de lemmatisation dans chacun des deux programmes de transformation écrits en Turbo-Pascal sur IBM-PC.



* procédure et fichier inclus ajoutés pour le transfert du fichier des contraintes de lemmatisation

2.1.1.2. Version intégrable à O_DI_LE

2.1.1.2.1. Adaptation du code

Dans cette phase, notre objectif était d'extraire de la version brute de PILAF, les deux parties de traitement devant constituer les deux points d'entrée du lemmatiseur (cf. chapitre II paragraphe 3.1.4.

Dans la version brute de PILAF, toute chaîne de caractères subit un pré-traitement destiné à la normaliser. Il est également prévu des traitements annexes de correction de la chaîne initiale dans le cas où une occurrence ne peut être analysée (base ou désinence non trouvée), de trace d'exécution, etc.

Tous les traitements ne concourant pas au processus de lemmatisation proprement dit ne nous intéressent pas dans O_DI_LE. Nous supposons en effet que la chaîne de caractères sélectionnée dans le texte de l'application courante est bien formée (un blanc ou certains caractères de ponctuations et un blanc entre chaque mot en français) et que les occurrences sont correctement orthographiées. Si ça n'est pas le cas, il conviendra éventuellement d'adjoindre un module responsable de la normalisation et de la correction orthographique de chaînes de caractères, car ce n'est certainement pas le rôle du lemmatiseur d'effectuer ce travail.

En conséquence, toutes les procédures de PILAF concernant ces traitements complémentaires ont été supprimés de la version intégrable à O_DI_LE.

En outre, nous avons modifié la structure de sortie du lemmatiseur PILAF pour adapter la structure commune à tout lemmatiseur intégrable à O_DI_LE définie au paragraphe 3.1.3. du chapitre II. Plus précisément, nous avons avons choisi de conserver l'ancienne structure de sortie adaptée `Hphlem`, décrite à la page 116, comme structure de travail et cela pour deux raisons :

- elle permet déjà de construire un fichier texte de résultats de lemmatisation factorisés que nous reprendrons pour alimenter en partie la structure `StructAffich` d'affichage à la demande des résultats de lemmatisation dans O_DI_LE,
- elle permet également de construire facilement le fichier texte `StructSortieOdile` définie comme structure de sortie commune.

La suppression pure et simple de l'ancienne structure de sortie, `Hphlem`, aurait nécessité des modifications importantes du code des procédures d'analyse, alors que la solution choisie reporte l'adaptation en fin d'analyse de la chaîne d'entrée. En effet, elle se résout facilement par l'adjonction des deux procédures `EcrStructSortieLem` et `CreatFText` en fin de la

procédure de génération de lemmes `GenerLem` de l'unité `PL_GenerLem`.

Enfin, la présence de la variable `chaine` en paramètre du point d'entrée `PL_Lemmatiser` a nécessité la modification des nombreuses procédures de PILAF appelées par ce point d'entrée et qui manipulaient la variable `chaine`. Il a en effet fallu leur ajouter le paramètre `chaine` qui n'y figurait pas puisque cette variable était globale (`chaineinit`).

2.1.1.2.2. Normalisation des noms d'unités et de fichiers

Les noms d'unité du module WinTool™ sont préfixés par "WT_", les noms d'unité des modules d'interface sont préfixés par "I_" ("I_Lem_" pour l'interface ODILE - Lemmatiseurs, "I_WT_" pour l'interface ODILE - WinTool), les noms d'unité du module ODILE sont préfixés par "O_".

Il était donc tout à fait logique que les noms des unités composant le module de lemmatisation PILAF intégrable à ODILE soient normalisés selon le même principe. Elles comportent donc toutes le préfixe "PL_" pour PILAF.

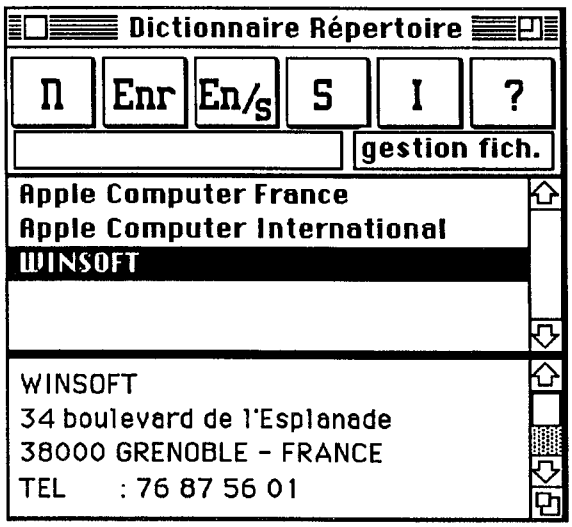
Nous en avons profité pour normaliser également le nom des fichiers contenant ces unités :
- extension ".p" pour les fichiers contenant des unités de code Pascal,
- extension ".h" pour les fichiers contenant des unités de déclaration.

2.1.2. Adaptation de WinTool™

Une nouvelle interface utilisateur de WinTool™ a été définie pour les besoins d'ODILE et également dans l'intérêt de Winsoft qui souhaitait "donner un coup de jeune" à WinTool™ et en même temps supprimer certaines limitations qui avaient été introduites au moment du développement initial du produit (ouverture simultanée de bases limitée à deux, fonction de création de bases WinTool™ non disponible dans l'accessoire de bureau mais localisée dans une application indépendante DocteurBase®). Il est désormais possible, dans WinTool™ 2.0, de créer de nouvelles bases et d'avoir plus de deux bases ouvertes en même temps.

Les fonctionnalités de WinTool™ 2.0, la nouvelle version de WinTool™, sont donc globalement inchangées. Seule sa présentation change : après son installation et l'ouverture d'un fichier (ou base), WinTool™ affiche une fenêtre de dialogue. Cette fenêtre duplique, sous forme de boutons et de menu fugitif, les commandes correspondant aux articles du menu WinTool™. Pour pouvoir intégrer WinTool™ à ODILE, il a en effet été nécessaire de transformer son interface pour répondre à la contrainte du standard Macintosh déjà évoquée au

paragraphe 1.1.1 du chapitre II et qui interdit aux accessoires de bureau d'installer plus d'un menu.



Dans sa nouvelle version, WinTool™ est converti en THINK Pascal™ 3.01. Cet environnement de programmation, que nous avions choisi pour développer ODILE, a été soumis à la société WinSoft. Celle-ci a rapidement été convaincue des avantages qu'il procurait pour le développement et la maintenance des applications. La réalisation de WinTool™ 2.0 et la mise en place des points d'entrée décrits au chapitre II paragraphe 3.2.1. ont été de la responsabilité de WinSoft.

2.2. Fonctions propres d'ODILE

Une première fonction d'ODILE consiste à analyser la chaîne sélectionnée dans le texte de l'application hôte. Elle vise à déterminer le nombre d'occurrences de la chaîne sélectionnée pour limiter éventuellement son nombre au nombre maximal d'occurrences traitable défini comme un paramètre d'ODILE.

```
fonction O_DeterminerOcc(Chaine: Chaine[255]; ParamOcc: Entier;
var HTabOcc: TabOccHdl; var NbOcc: Entier): Entier;
```

où

- `Chaine: Chaine[255]` est la chaîne de caractères sélectionnée,
- `ParamOcc: Entier` est le paramètre nombre d'occurrences maximal (menu "Préférences..."),
- `HTabOcc: TabOccHdl` est une poignée référençant le tableau dynamique des occurrences déterminées. `NbOcc: Entier` est le nombre d'occurrences de ce tableau, c'est-à-dire le nombre d'occurrences déterminées dans la chaîne sélectionnée.

Cette fonction découpe la chaîne sélectionnée (`Chaine`) en occurrences. Une occurrence est toute suite de caractère différente d'un caractère de ponctuation (",", ";", ":", "!", "?") comprise entre deux caractères séparateurs d'occurrences qui sont :
- le caractère "blanc" (" ")
- les caractères de ponctuation,
- le caractère apostrophe (" ' ").

L'apostrophe joue un rôle particulier puisque c'est à la fois un séparateur et un caractère appartenant à une occurrence lorsqu'on le rencontre comme séparateur à droite de cet occurrence.

Exemple :
Soit la chaîne "l'avion", alors la fonction détermine les deux occurrences "l'" et "avion".

La fonction rend un résultat entier, qui vaut 0 si le nombre d'occurrences déterminé `NbOcc` est inférieur ou égal au nombre d'occurrences maximal `ParamOcc` et qui est négatif sinon.

La coopération efficace entre les logiciels PILAF et WinTool™ nécessite l'existence de procédures spécifiques d'ODILE évoquées au paragraphe 2.1.1. du chapitre II qui décrit le principe de génération des clés de recherche. Elles sont au nombre de trois.

Les procédures `O_CombinerFetL` et `O_FormesSeules` fabriquent des clés potentielles de recherche dans un dictionnaire usuel WinTool™.

La procédure `O_CombinerFetL` fabrique des clés de recherche en combinant les formes et les lemmes de la structure `HTabStructOdile`, la structure propre d'ODILE chargée à partir de la structure résultat de la lemmatisation.

```
procedure O_CombinerFetL (HTabStructOdile: TabStructOdileHdl;
NbElem: Entier; var HTabCle: TabCleHdl; var NbCle: Entier);
```

où
- `HTabStructOdile: TabStructOdileHdl` est une poignée sur un tableau dynamique constitué des résultats de lemmatisation de la chaîne d'entrée utiles pour ODILE. `NbElem: Entier` est le nombre d'éléments de ce tableau.
- `HTabCle: TabCleHdl` est le tableau dynamique des clés de recherches potentielles calculées par ODILE. Il est destiné à alimenter la structure graphique (liste statique) des clés de recherche proposées à l'utilisateur. `NbCle: Entier` est le nombre d'éléments de ce tableau.

La procédure `O_FormesSeules` fabrique une seule clé de recherche de la forme (forme){blanc(forme)}* en concaténant toutes les formes de la structure `HTabOcc` déterminée par la fonction propre d'ODILE `O_DeterminerOcc`.

```
procedure O_FormesSeules (HTabOcc: TabOccHdl; NbOcc: Entier;
var HTabCle: TabCleHdl; var NbCle: Entier);
```

où

- `HTabOcc: TabOccHdl` est une poignée sur le tableau dynamique des occurrences (ou formes) déterminée par la fonction propre d'ODILE `O_DeterminerOcc` à partir de la chaîne d'entrée. `NbOcc` est le nombre d'occurrences de cette structure.
- `HTabCle: TabCleHdl` a la même définition que dans la fonction précédente, mais lorsque cette structure est alimentée par la fonction `O_FormesSeules` elle ne contient qu'un élément. `NbCle: Entier` est le nombre d'élément de ce tableau.

La procédure `O_Serrurier`, recherche des clés potentielles déjà calculées, dans un lexique personnel, en fonction de la chaîne d'entrée.

```
procedure O_Serrurier (Chaine: Chaine[255]; NumBase: Entier;
NbCle: Entier; var HTabCle: TabCleHdl; var NouvNbCle: Entier):
Entier;
```

où

- `Chaine: Chaine[255]` est la chaîne de caractères traitée, c'est-à-dire la chaîne sélectionnée éventuellement limitée au nombre maximal d'occurrences traitables,
- `NbCle: Entier` est le nombre d'éléments du tableau des clés de recherche potentielles `HTabCle` à l'entrée de la procédure,
- `HTabCle: TabCleHdl` est le tableau des clés de recherches potentielles ; il est augmenté des clés trouvées dans le lexique personnel identifié par `NumBase`,
- `NouvNbCle: Entier` est le nouveau nombre d'éléments du tableau des clés de recherche en sortie de la procédure.

La procédure rend un entier négatif si elle a rencontré un problème pour accéder au lexique personnel, 0 si tout s'est bien passé.

Concernant la gestion des clés personnelles, nous avons évoqué la possibilité de charger ces fichiers, de taille raisonnable a priori, dans des tables en mémoire centrale et d'effectuer les recherches par dichotomie (chapitre II paragraphe 4.1.2) ; dans ce cas, toutes les procédures de gestion de ces fichiers sont à écrire.

Pour une question de facilité et de rapidité de mise en œuvre, nous avons opté, dans la première version d'ODILE, pour la gestion de ces fichiers au moyen des procédures de WinTool™. Dans un premier temps, les fichiers des clés utilisateurs auront un format WinTool™, c'est-à-dire une organisation séquentielle indexée avec index en B-arbre, même si cela ne se justifie pas vraiment.

2.3. Interface utilisateur

La programmation d'une interface utilisateur d'ODILE nécessite la connaissance des procédures de la ToolBox, la boîte à outils du Macintosh. L'apprentissage de la ToolBox a été progressif.

Dans un premier temps, nous avons essayé de définir une interface utilisateur autour du module d'analyse morphologique de PILAF. Cela nous a permis de mettre en œuvre la logique des applications Macintosh vue au paragraphe 1.2.1.2 de ce chapitre, et de découvrir les nombreuses routines de la ToolBox.

Cela a également été l'occasion d'expérimenter ResEdit, le logiciel de définition des ressources sur Macintosh. Une ressource définit généralement un "morceau" d'interface utilisateur (menu, fenêtre, dialogue, alerte, icône, ...) [FONT88]. Les ressources utilisées par une application sont stockées séparément de son code pour une plus grande souplesse et une plus grande facilité de maintenance.

Les ressources sont classées logiquement en différents types de ressources (MENU, DLOG, WIND, ALRT, ICON,...). Chaque ressource d'un type donné est identifié par un numéro de type, son "ressource ID". Dans un fichier de ressource donné, une ressource est donc identifiée par son type et son "ID".

Une ressource d'une application est accédée et manipulée par certaines procédures de la ToolBox auxquelles on fournit son identificateur en paramètre. ResEdit permet la création et la modification d'un fichier de ressources dans un état exploitable par l'application qui va les utiliser.

En THINK Pascal™, une application est désignée sous le terme de projet. La désignation du fichier de ressources utilisable par l'application se fait par la commande de définition des options d'exécution ("Run Options").

Une meilleure connaissance du concept d'application Macintosh nous a permis, dans un

deuxième temps, de comprendre le fonctionnement des accessoires de bureau.

Les accessoires de bureau sont en quelque sorte des applications particulières puisqu'ils sont disponibles à tout moment à partir du "menu pomme". En fait, ils sont toujours subordonnés à une application hôte ; leur bon fonctionnement dépend de la bonne conception de l'application à partir de laquelle ils sont appelés. Examinons comment cela est possible.

Nous avons vu au paragraphe 1.2.1.2. que toute application Macintosh contient une procédure de traitement des événements en fonction de leur nature (HandleEvent). Cette procédure doit en particulier traiter l'événement "InSysWindow" qui signifie que l'utilisateur a cliqué dans une fenêtre système donc dans la fenêtre d'un accessoire de bureau puisque les accessoires de bureau sont chargés dans le système du Macintosh.

Le traitement de cet événement consiste en l'appel de la routine "SystemClick" de la ToolBox. Cette procédure remplit la structure de communication adéquate avant de "passer la main" au système qui dirige l'exécution vers l'entrée Open, Control, ou Close de l'accessoire de bureau.

A la suite de cet apprentissage, nous avons amorcé le développement de l'interface utilisateur d'ODILE "vide", c'est-à-dire la partie Présentation de l'accessoire pour reprendre le vocabulaire PAC.

2.4. Architecture logicielle

La structuration logicielle d'ODILE selon le modèle PAC, étudiée au chapitre II, s'est avérée trop lourde à mettre en oeuvre avec le langage choisi.

A chaque objet PAC doit en effet correspondre une unité Pascal. La partie implémentation de chaque unité contient trois groupes de procédures (ou fonctions) : des procédures traitant la partie abstraction de l'objet, des procédures traitant la partie présentation de l'objet et des procédures traitant la partie contrôle de l'objet.

La partie implémentation de chaque unité ne contient que les procédures du contrôle de l'objet puisque les objets ne peuvent communiquer que par l'intermédiaire de leur contrôle. De même, la communication entre les parties présentation et abstraction d'un objet passe par le contrôle de cet objet. Il s'ensuit des appels de procédures en cascade simplement pour gérer la communication.

Pour expérimenter ce modèle de structuration du Système Interactif ODILE, il aurait, d'une part, peut-être fallu choisir un langage de programmation à objets, plutôt qu'un langage procédural comme THINK Pascal™.

D'autre part, le manque d'outils adaptés au développement des applications selon ce modèle (générateurs d'interfaces, squelettes d'applications) alourdit ce développement même si de tels outils sont en cours d'étude dans les laboratoires (APPEX [COUT88], générateur d'interfaces [NIGAY90]). Cela anéantit pratiquement les avantages que la structuration PAC apporte au niveau de l'évolution, donc de la maintenance, du système interactif.

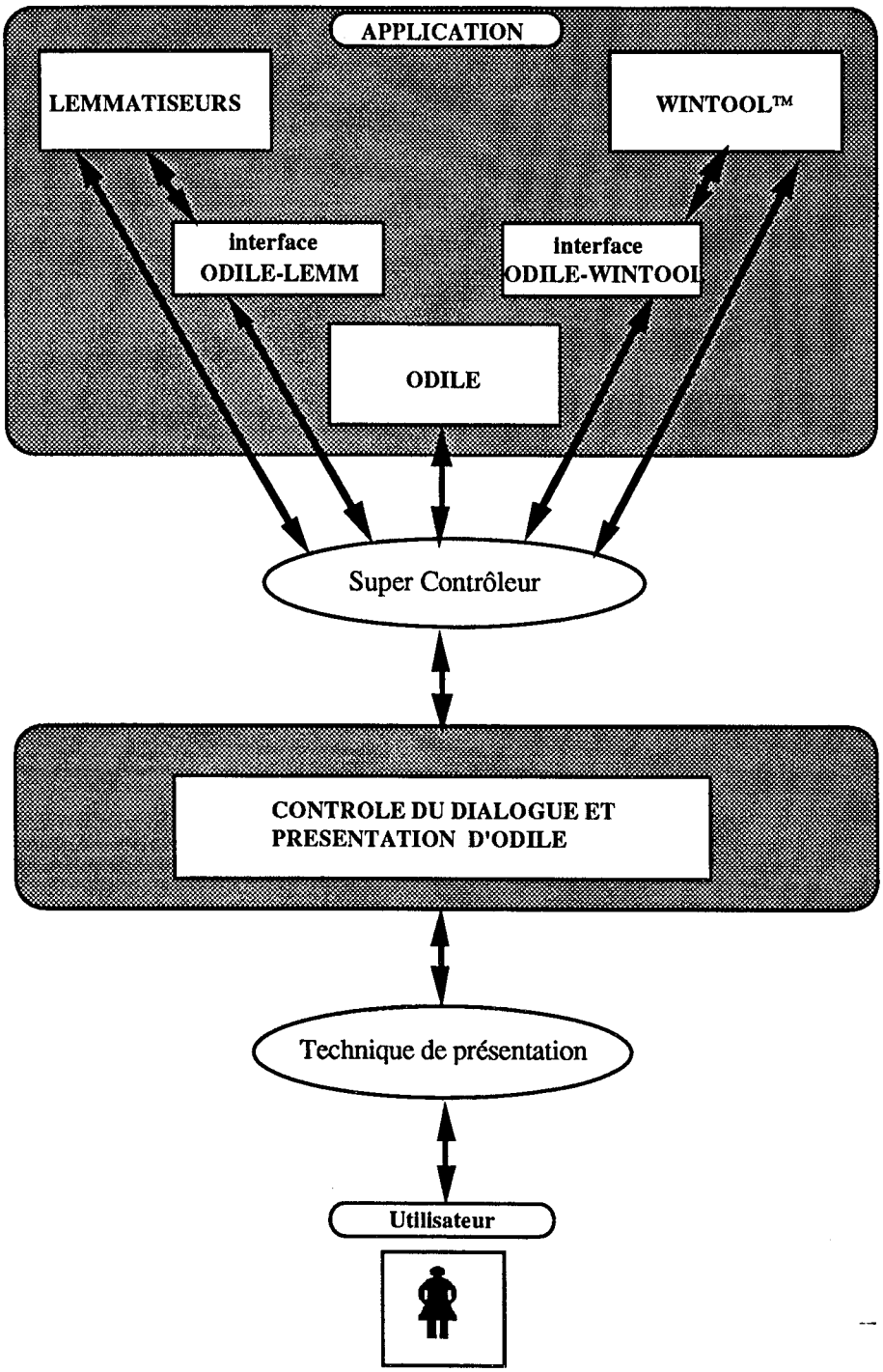
Nous n'avons donc pas poussé plus loin cette expérience, mais nous avons gardé du modèle PAC quelques aspects intéressants. Plus précisément, nous avons conservé l'architecture générale proposée par le modèle Nouv-PAC (parties Application, Super Contrôleur, Contrôle du Dialogue et Présentation, et Technique de Présentation).

La partie Contrôle du dialogue et Présentation, qui décrit le comportement su système interactif ODILE, n'est pas ici organisée en un hiérarchie d'objets PAC, mais nous avons voulu établir une séparation nette dans le code entre la partie présentation et la partie abstraction.

Nous avons distingué par des commentaires appropriés les parties de code destinées aux traitements graphiques et celles qui sont destinées aux calculs et mises à jour des données internes. Nous avons prévu une procédure dédiée à la gestion des messages d'alerte et d'erreur. Cette solution est préférable à celle qui consiste à construire le message directement dans la procédure qui en a besoin.

Enfin nous avons distingué la partie du code chargée de l'analyse et de la distribution des événements (technique de présentation) en créant une "unit" spécifique.

Le schéma qui suit décrit l'architecture retenue pour le système interactif ODILE.



Architecture du Système Interactif ODILE

Détaillons le contenu de chacun des modules.

Les modules LEMMATISEURS, WINTOOL™, et ODILE regroupent des fonctionnalités internes. Chacun de ces modules regroupe un certain nombre d'unités qui décrivent sa compétence ; ils appartiennent à la partie application du système interactif :

- le module LEMMATISEURS est responsable de la lemmatisation des occurrences,

- le module WINTOOL™ est responsable de la recherche dans les dictionnaires usuels et de la gestion des dictionnaires (dictionnaires usuels et lexiques personnels),
- le module ODILE assure la fonction de contrôle du nombre d'occurrences de la chaîne de caractères sélectionnée et les fonctions de fabrication des clés de recherche potentielles.

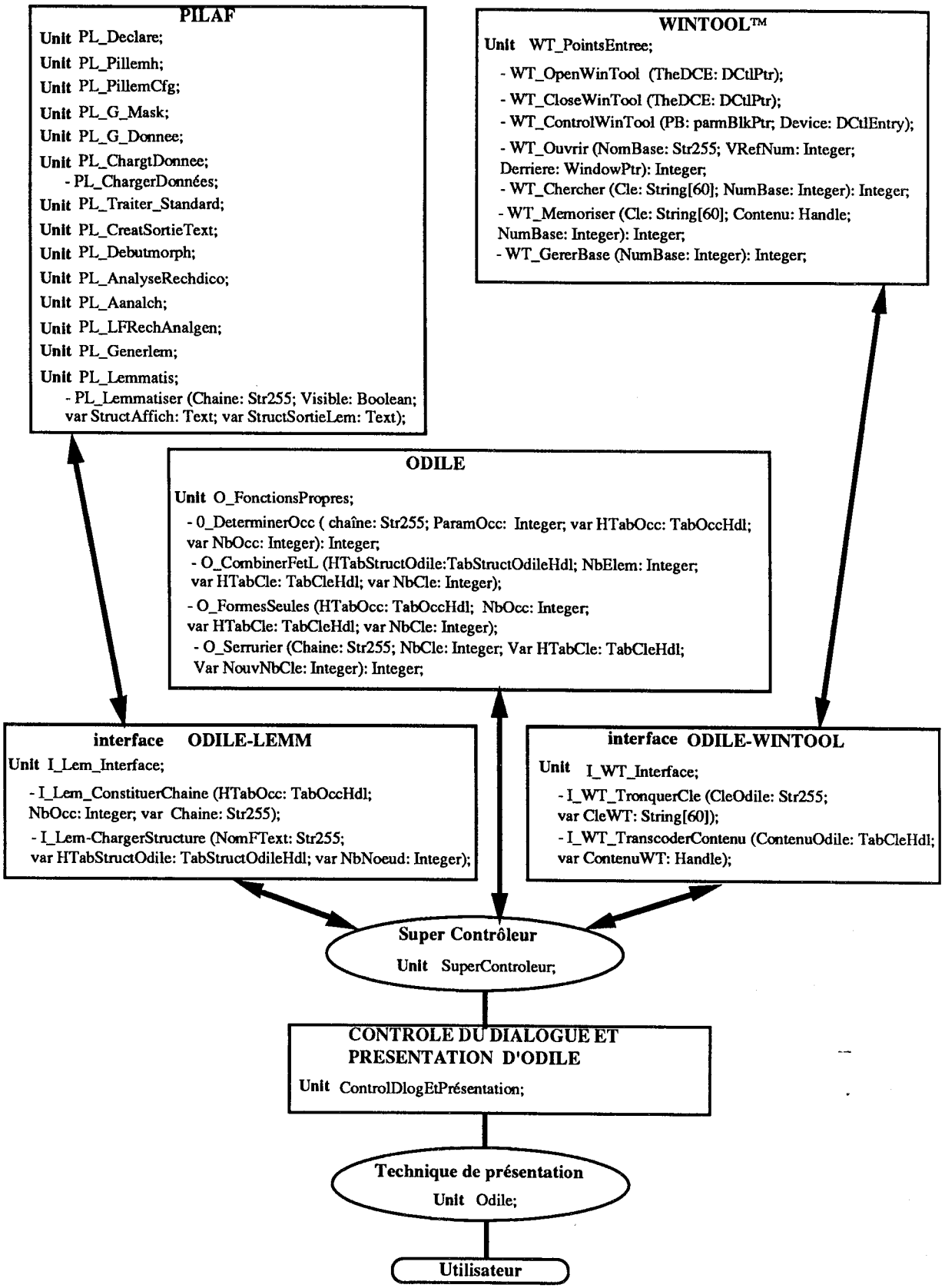
Les modules interface ODILE-LEMM et interface ODILE-WINTOOL assurent les changements de formalismes nécessaires à la communication entre le Système Interactif ODILE et Lemmatiseurs d'une part, et le Système Interactif ODILE et WinTool™ d'autre part, selon les spécifications décrites au chapitre II paragraphes 3.1.5. et 3.2.3..

Le module SUPER CONTROLEUR est un module tampon. Il est constitué de procédures qui appellent des procédures des modules LEMMATISEURS, WINTOOL™ et ODILE. Son intérêt est de rendre transparents au système interactif des changements éventuels dans les modules de l'Application. En cas d'intégration de plusieurs lemmatiseurs dans ODILE, il permet également de diriger les traitements vers les procédures du lemmatiseur choisi en paramètre : il joue en quelque sorte le rôle d'aiguillage.

Le module CONTROLE DU DIALOGUE ET PRESENTATION D'ODILE reçoit les événements en provenance de l'utilisateur, les traite en appelant les fonctions adaptées des modules de l'abstraction et renvoie à l'utilisateur une image du système mise à jour (par appel de procédures de la ToolBox).

Le module TECHNIQUE DE PRESENTATION est responsable de l'analyse des événements et de leur distribution vers les objets interactifs concernés (appel de la procédure du module CONTROLE DU DIALOGUE ET PRESENTATION D'ODILE adaptée pour traiter cet événement). D'un point de vue logique, on peut considérer que ce module englobe les procédures de la ToolBox.

Dans le schéma qui suit, nous donnons une vue plus détaillée de chaque module en faisant apparaître les unités appartenant à chaque module.


PILAF
Unit PL_Declare;
Unit PL_Pillemh;
Unit PL_PillemCfg;
Unit PL_G_Mask;
Unit PL_G_Donnee;
Unit PL_ChargtDonnee;
- PL_ChargerDonnées;
Unit PL_Traiter_Standard;
Unit PL_CreatSortieText;
Unit PL_Debutmorph;
Unit PL_AnalyseRechdico;
Unit PL_Aanalch;
Unit PL_LFRechAnalgen;
Unit PL_Generlem;
Unit PL_Lemmatis;
- PL_Lemmatiser (Chaine: Str255; Visible: Boolean; var StructAffich: Text; var StructSortieLem: Text);
WINTOOL™
Unit WT_PointsEntree;
- WT_OpenWinTool (TheDCE: DCtlPtr);
- WT_CloseWinTool (TheDCE: DCtlPtr);
- WT_ControlWinTool (PB: parmBlkPtr; Device: DCtlEntry);
- WT_Ouvrir (NomBase: Str255; VRefNum: Integer; Derriere: WindowPtr): Integer;
- WT_Chercher (Cle: String[60]; NumBase: Integer): Integer;
- WT_Memoriser (Cle: String[60]; Contenu: Handle; NumBase: Integer): Integer;
- WT_GererBase (NumBase: Integer): Integer;
ODILE
Unit O_FonctionsPropres;
- 0_DeterminerOcc (chaîne: Str255; ParamOcc: Integer; var HTabOcc: TabOccHdl; var NbOcc: Integer): Integer;
- O_CombinerFetL (HTabStructOdile:TabStructOdileHdl; NbElem: Integer; var HTabCle: TabCleHdl; var NbCle: Integer);
- O_FormesSeules (HTabOcc: TabOccHdl; NbOcc: Integer; var HTabCle: TabCleHdl; var NbCle: Integer);
- O_Serrurier (Chaine: Str255; NbCle: Integer; Var HTabCle: TabCleHdl; Var NouvNbCle: Integer): Integer;
interface ODILE-LEMM
Unit I_Lem_Interface;
- I_Lem_ConstituerChaine (HTabOcc: TabOccHdl; NbOcc: Integer; var Chaine: Str255);
- I_Lem-ChargerStructure (NomFText: Str255; var HTabStructOdile: TabStructOdileHdl; var NbNoeud: Integer);
interface ODILE-WINTOOL
Unit I_WT_Interface;
- I_WT_TronquerCle (CleOdile: Str255; var CleWT: String[60]);
- I_WT_TranscoderContenu (ContenuOdile: TabCleHdl; var ContenuWT: Handle);
Super Contrôleur
Unit SuperControleur;
CONTROLE DU DIALOGUE ET PRESENTATION D'ODILE
Unit ControlDlogEtPrésentation;
Technique de présentation
Unit Odile;
Utilisateur

3. Perspectives et bilan

3.1. Amélioration des performances d'ODILE

L'amélioration des performances d'ODILE est directement liée à l'amélioration des performances du lemmatiseur, et plus précisément à la modification de la structure de son dictionnaire.

La structure évoquée au paragraphe 4.2. du chapitre II représente une voie intéressante de recherche dans cette optique. Cependant, une étude approfondie de cette structure en vue de son adaptation à PILAF demandait d'y consacrer du temps dont nous ne disposions pas compte tenu de nos priorités. Nous souhaitions en effet réaliser un prototype d'ODILE afin d'en évaluer l'intérêt, même si ses performances sont très moyennes.

Nous avons trouvé un compromis permettant à la fois de réduire la taille du dictionnaire tout en le maintenant modifiable et de diminuer dans certains cas le temps d'accès à ses éléments. Pour cela, nous avons adapté au Macintosh la structure étudiée au LGI par Eric LOPEZ (cf paragraphe 4.1.3.1. du chapitre II) et en partie implémentée sur IBM PC. Les dictionnaires existants dans la nouvelle structure sur IBM-PC ont été transcodés pour être exploitable sur Macintosh.

L'implémentation s'appuie sur un module d'interface constitué de différentes procédures d'accès au dictionnaire. Cet interfaçage a l'intérêt de rendre transparente aux modules de traitement de PILAF toute modification ultérieure éventuelle de la structure du dictionnaire : seules les procédures de l'interface doivent être adaptées.

L'adaptation de la nouvelle structure au Macintosh a été découpée en trois phases : les deux premières phases sont réalisées, la troisième est en cours.

La première phase a consisté à porter le module d'interfaçage entre les programmes PILAF et la structure du dictionnaire sur Macintosh. Dans la deuxième phase, nous avons porté le dictionnaire dans sa nouvelle structure de l'IBM PC sur Macintosh. Ce dictionnaire contient 35000 entrées. Nous ne pouvions pas compter utiliser les fichiers de données dont nous disposions jusqu'alors sur Macintosh, même si ces fichiers n'ont pas subi de modification de structure, pour la simple raison qu'ils travaillent avec un dictionnaire de bases de 3000 entrées et qu'il y a alors un risque pour que les données ne soient pas en phase avec le nouveau dictionnaire. Nous avons donc également dû porter tous les fichiers de données travaillant avec le dictionnaire de 35000 bases.

Pour ce faire, nous avons utilisé les programmes de conversion développés par les participants au projet MacPILAF après les avoir modifiés pour les adapter à la nouvelle structure du dictionnaire de bases. Nous avons en particulier modifié la procédure "Transformer_tabmaitre" dans le sens de conversion Interne-Externe et les fichiers inclus "UTIL_TM.001" et "UTIL_TABM.001" dans le sens de conversion Externe-Interne (cf. schéma page 118).

Enfin, la troisième étape, non encore réalisée, consistera à remplacer les différentes primitives d'accès aux éléments de dictionnaire par l'appel aux procédures du module d'interface.

3.2. Travail restant sur la première version

A l'heure actuelle, certaines fonctionnalités prévues dans notre prototype n'ont pas encore été réalisées.

La fonction Serrurier n'est pas programmée, mais toutes ses spécifications sont décrites et il sera facile de la mettre en place.

La partie de l'interface utilisateur non implémentée permettant d'utiliser le lemmatiseur en indépendant (article "Lemmatiseur...") notamment pour la modification des données et du dictionnaire du lemmatiseur sera plus longue à mettre en œuvre. En effet, elle fait appel à des fonctionnalités décrites dans des modules particuliers conséquents de PILAF, les modules PILDIC et PILPAR, et nécessite la mise en place de niveaux d'utilisation. Cette partie reste à étudier.

Il reste également à implémenter la modification des paramètres d'utilisation (article "Préférences...").

L'interface utilisateur de notre prototype mérite aussi quelques compléments. Au cours de la réalisation, nous avons entrevu des améliorations possibles, parmi lesquelles :

- la suppression du dialogue d'ouverture d'un dictionnaire de lemmatisation à l'installation d'ODILE ; un dictionnaire sera choisi par défaut,
- la modification de la fonction de mémorisation de clés dans un lexique personnel ; seules les clés sélectionnées dans la liste statique des clés seront mémorisées (et non pas toutes) ; cela signifie que, pour cette opération, on devra pouvoir sélectionner plusieurs clés à la fois dans la liste,
- la modification de la commande "Ajouter" ; on pourra éditer une clé de la liste statique dans la zone de saisie et la modifier pour ensuite l'ajouter, plutôt que de saisir toute la chaîne de

caractères de la clé à ajouter,
- l'ajout de messages pour informer l'utilisateur de l'état du système,
- l'ajout d'une aide en ligne.

L'objectif principal a fait passer au second rang un aspect d'étude important : l'amélioration de la structure du dictionnaire PILAF. Il n'en reste pas moins qu'il est souhaitable de continuer l'étude de la solution ébauchée à la fin du chapitre II pour voir notamment si l'organisation proposée est facilement adaptable à PILAF et répond bien a notre attente en termes de gain de place et d'amélioration du temps de traitement global.

3.3. Extensions envisageables

Après les améliorations nécessaires à l'obtention d'un prototype plus représentatif de ce que sera ODILE, on peut envisager certaines extensions, qui permettraient peut-être d'en faire un produit.

Une extension évidente est d'exploiter la caractéristique extensible d'ODILE et d'intégrer d'autres lemmatiseurs et en particulier un lemmatiseur tiré de l'analyseur ATEF pour évaluer son comportement dans ODILE par rapport à celui de PILAF. On pourra également définir une autre version d'ODILE utilisant l'outil dictionnaire WinFile™ pour exploiter des dictionnaires de formats divers et dont les enregistrements possèdent plusieurs champs clé. On pourra imaginer alors de consulter des dictionnaires dont les enregistrements ont pour clé, non seulement une chaîne de caractères, mais aussi la catégorie syntaxique de cette chaîne qui permet de discriminer les entrées homographes du dictionnaire (par exemple "ferme" : adjectif, adverbe, nom commun ou verbe). Dans ce contexte, il conviendra d'aménager ODILE pour déterminer la catégorie syntaxique de chacune des entrées potentielles fabriquées.

Dans le contexte d'une application d'ODILE à la traduction de petits volume de documentation, il conviendra d'adjoindre un module pour déterminer les équivalents fléchis pouvant correspondre aux termes à traduire.

On pourra également étudier l'intérêt d'un mode automatique de recherche des clés potentielles dans un dictionnaire usuel. Dans cette optique, il faudra définir les modifications à apporter à l'interface utilisateur d'ODILE après étude des solutions envisagées. Il faudra également évaluer les conséquences de ce mode de fonctionnement sur l'outil dictionnaire et mettre en place des options permettant à l'utilisateur de spécifier un ordre de recherche de ces clés.

Si l'on approfondit cette idée, on peut envisager d'introduire un filtre syntaxique chargé d'éliminer les clés potentielles aberrantes de la liste des clés automatiquement recherchées.

Conclusion

La plus grande part de ce travail a consisté en un travail de spécification d'un outil interactif d'aide linguistique sur micro-ordinateur dont on ne savait pas très bien au départ ce qu'il serait. Il a fallu clarifier nos idées, définir nos besoins pour énoncer des principes de fonctionnement du futur logiciel.

La mise au point des spécifications de l'interface utilisateur d'un système interactif est assez longue. Pour notre part, nous avons exploré diverses solutions abandonnées ensuite en raison de contraintes techniques ou bien en raison du manque d'ergonomie du système final.

En règle générale, toute modification de l'interface entraîne sa réévaluation complète pour détecter une éventuelle incohérence devant être corrigée. Nous avons constaté que ce processus itératif est inévitable et que les règles énoncées pour faciliter la conception des systèmes interactifs servent plus de guide que de lois absolues. Il faut constamment effectuer un subtil dosage des différents principes pratiques ergonomiques. Ces règles n'en restent pas moins très utiles dans la mesures où elles permettent d'éviter des maladresses voire des erreurs de conception.

Le besoin de structurer notre logiciel nous a conduit à l'étude du modèle PAC de structuration des systèmes interactifs et nous a convaincu de l'intérêt qu'un tel modèle présente pour l'évolution et la maintenance des systèmes interactifs. Malheureusement, les outils permettant une mise en œuvre aisée d'un tel modèle ne sont pas encore opérationnels. Il faudra donc suivre de près les progrès et les résultats obtenus dans ce domaine d'étude afin d'exploiter au plus vite les avantages d'un tel modèle.

Nous pouvons regretter que le temps de portage du lemmatiseur sur Macintosh ainsi que le temps de développement de l'interface utilisateur ait été sous-estimé. En effet, l'apprentissage du Macintosh, et plus précisément la compréhension des mécanismes qui régissent son système d'exploitation, et l'apprentissage de sa boîte à outils ont demandé un temps considérable.

En conséquence, au moment où nous rédigeons ce mémoire, la réalisation de notre outil n'a pas pu être menée à son terme : l'adaptation des différents logiciels, la réalisation des diverses interfaces et l'intégration du lemmatiseur PILAF sont effectives, mais l'intégration de WinTool™ n'a pu être faite. En effet, en raison de contraintes de temps, la version de WinTool™ intégrable à ODILE n'a pas pu nous être livrée pour la date voulue. Dans les mois qui viennent, nous comptons poursuivre notre tâche et nous espérons atteindre notre objectif principal qui est de réaliser un prototype, même modeste, d'un intégrateur de logiciels dans le

domaine du traitement de la langue naturelle.

En revanche, le portage du coeur du lemmatiseur extrait de PILAF sur Macintosh est effectif et c'est un point positif. Il doit permettre son utilisation dans d'autres applications sur ce matériel.

La préparation de ce mémoire nous a permis de découvrir un domaine d'application de l'informatique, vaste et jusque là encore totalement inconnu, la linguistique. Il nous a également familiarisé avec un domaine de l'informatique que nous n'avions pas encore expérimenté, la micro-informatique, et nous a plus précisément apporté la connaissance d'une machine particulière, le Macintosh.

Le double objectif de ce projet, à savoir concevoir et réaliser un logiciel qui soit à la fois modulaire, paramétrable, mettant en œuvre une technique de structuration logicielle relativement nouvelle (PAC) et en accord avec des critères d'ergonomie et de performances, était ambitieux. Il nous a permis de confronter deux mondes différents que sont les laboratoires de recherche et les sociétés commercialisant des logiciels et de nous apercevoir que certains de leurs vues, liées à des objectifs différents, ne sont pas toujours compatibles.

Dans sa première version, l'outil est une ébauche qui, nous l'espérons, permettra de voir quels sont les aspects à privilégier dans les versions suivantes, parmi l'intégration, la propreté, la généricité et l'ergonomie. Nous avons en effet constaté que ces aspects sont en contradiction dans la mesure où, par exemple, l'objectif de propreté conduit à un fonctionnement de l'outil dont l'ergonomie est contestable.

Dans un futur proche, une nouvelle version doit être étudiée dans le cadre d'un projet de DEA concernant les outils d'aide au traducteur sur micro-ordinateur. Elle devrait apporter certaines améliorations déjà envisagées dans ce mémoire et sans doute aussi des solutions aux problèmes mis en évidence par le prototype.

Dans le cadre d'un projet du Réseau Francophone des Industries de la Langue mené en coopération avec l'Institut d'Etude et de Recherche pour l'Arabisation (IERA) de Rabat (Maroc) et l'équipe TRILAN du LGI, il est enfin prévu de faire prochainement un test avec 15000 termes (en commerce, comptabilité et banque) tirés de la base terminologique de l'IERA. Nous espérons à cette occasion voir si un tel outil serait réellement utilisable par des traducteurs (en principe occasionnels, mais formés dans les deux langues) de français en arabe.

Annexes

ANNEXE 1 - Fichier des contraintes et fichier des classes lexicales de PILAF sous format externe standard.

Dans le fichier des contraintes de lemmatisation,

- l'information NUMERO représente le rang de la contrainte dans le fichier,
- l'information CL. ORIG représente le rang dans le fichier des classes lexicales de la classe lexicale d'origine,
- l'information NUM.CL.SUBST. représente le rang dans le fichier des classes lexicales de la classe lexicale du lemme à générer pour la classe d'origine donnée,
- l'information MASQUE VAR. représente les variables morphologiques du lemme à générer pour la classe d'origine donnée,

********** FICHIER DES CONTRAINTES DE LEMMATISATION **********

NUMERO	CL.ORIG.	NUM.CL.SUBST.	MASQUE VAR.
(1)	(4)	(4)	('sin')
(2)	(3)	(3)	('')
(3)	(5)	(5)	('sin mas')
(4)	(6)	(6)	('sin')
(5)	(7)	(7)	('')
(6)	(8)	(8)	('sin mas')
(7)	(10)	(10)	('')
(8)	(11)	(10)	('')
(9)	(12)	(10)	('')
(10)	(13)	(10)	('')
(11)	(14)	(10)	('')
(12)	(15)	(10)	('')
(13)	(16)	(16)	('')
(14)	(17)	(17)	('')
(15)	(18)	(18)	('')
(16)	(19)	(19)	('')
(17)	(20)	(20)	('')
(18)	(21)	(21)	('')

(19)	(22)	(22)	('')
(20)	(23)	(23)	('')
(21)	(24)	(24)	('')
(22)	(25)	(25)	('')
(23)	(26)	(26)	('')
(24)	(27)	(27)	('')
(25)	(28)	(28)	('')
(26)	(29)	(29)	('sin mas')
(27)	(30)	(30)	('')
(28)	(31)	(31)	('')
(29)	(32)	(10)	('')
(30)	(33)	(33)	('')
(31)	(34)	(34)	('')
(32)	(35)	(35)	('')
(33)	(36)	(36)	('sin mas')
(34)	(37)	(37)	('sin')
(35)	(38)	(38)	('')

Dans le fichier des classes lexicales,

- l'information NUMERO représente le rang de la classe lexicale dans le fichier,
- l'information CLASSE représente le code, limité à quatre caractères, de la classe lexicale,
- l'information REM représente le libellé en clair de la classe lexicale,
- les informations RTETG, RTETD, RSD, et CONFIG sont des informations exploitées par les procédures de PILAF chargées de la construction de graphes de dépendances en vue d'une analyse syntaxique. Nous ne nous y intéressons pas dans le cadre de la lemmatisation.

********** FICHIER DES CLASSES LEXICALES **********

NUMERO	CLASSE	RTETG	RTETD	REM	RSD	CONFIG
(1)	('clg')	(-1)	(-1)	('cat lex gauche ')	(F)	(255)

(2)	('cld')	(-1)	(-1)	('cat lex droite ') (F) (255)
(3)	('adv')	(119)	(27)	('adverbe ') (F) (255)
(4)	('subc')	(10)	(6)	('substantif commun ') (F) (255)
(5)	('detp')	(-1)	(41)	('determinant-pronom l') (F) (255)
(6)	('det')	(-1)	(10)	('determinant ') (F) (255)
(7)	('subp')	(98)	(7)	('substantif propre ') (F) (255)
(8)	('adjq')	(118)	(14)	('adjectif qualificati') (F) (255)
(9)	('vide')	(-1)	(-1)	(' ') (F) (255)
(10)	('infi')	(43)	(5)	('infinitif ') (F) (255)
(11)	('ppt')	(111)	(18)	('participe pre!3sent ') (F) (255)
(12)	('ppas')	(104)	(17)	('participe passe!3 ') (F) (255)
(13)	('verb')	(8)	(2)	('verbe ') (F) (255)
(14)	('xet')	(55)	(3)	('auxiliaire e!9tre ') (F) (255)
(15)	('xav')	(72)	(4)	('auxiliaire avoir ') (F) (255)
(16)	('pnt')	(-1)	(-1)	('point ') (F) (255)
(17)	('dpnt')	(124)	(30)	('deux points ') (F) (255)
(18)	('virg')	(-1)	(23)	('virgule ') (F) (6)
(19)	('coco')	(159)	(1)	('conjonc. coordinatio') (F) (8)
(20)	('prep')	(32)	(16)	('preposition ') (F) (2)
(21)	('locp')	(148)	(128)	('locution pre!3position')

				(F) (255)
(22)	('cocs')	(-1)	(12)	('conjonction subordin')
				(F) (255)
(23)	('locs')	(-1)	(129)	('locution conjonctive')
				(F) (255)
(24)	('padv')	(-1)	(25)	('pronom adverbial y e')
				(F) (255)
(25)	('prl')	(175)	(11)	('pronom relatif sauf ')
				(F) (2)
(26)	('prlc')	(176)	(158)	('pronom relatif conjo')
				(F) (255)
(27)	('ne')	(-1)	(24)	('negation ne ')
				(F) (255)
(28)	('pas')	(-1)	(26)	('ne!3gation pas ')
				(F) (255)
(29)	('ce')	(-1)	(63)	('pronom de!3monstratif ')
				(F) (255)
(30)	('pper')	(-1)	(19)	('pronom personnel ')
				(F) (255)
(31)	('phra')	(1)	(-1)	('phrase ')
				(V) (255)
(32)	('vers')	(11)	(-1)	('verbe ds subordonnee')
				(V) (255)
(33)	('pnp')	(121)	(20)	('pron.non personnel ')
				(F) (255)
(34)	('loca')	(-1)	(123)	('locution adverbiale ')
				(F) (255)
(35)	('ppf')	(-1)	(28)	('pron.personnel fort ')
				(F) (255)
(36)	('adji')	(-1)	(15)	('adjectif indefini ')
				(F) (255)
(37)	('chf')	(-1)	(-1)	('chiffre')
				(F) (255)
(38)	('date')	(-1)	(-1)	('date')
				(F) (255)
(39)	('')	(0)	(0)	(' ')
				(F) (255)
(40)	('')	(0)	(0)	(' ')
				(F) (255)

(41) (") (0) (0) (' ') (F) (255)

(42) (") (0) (0) (' ') (F) (255)

(43) (") (0) (0) (' ') (F) (255)

(44) (") (0) (0) (' ') (F) (255)

(45) (") (0) (0) (' ') (F) (255)

(46) (") (0) (0) (' ') (F) (255)

(47) (") (0) (0) (' ') (F) (255)

(48) (") (0) (0) (' ') (F) (255)

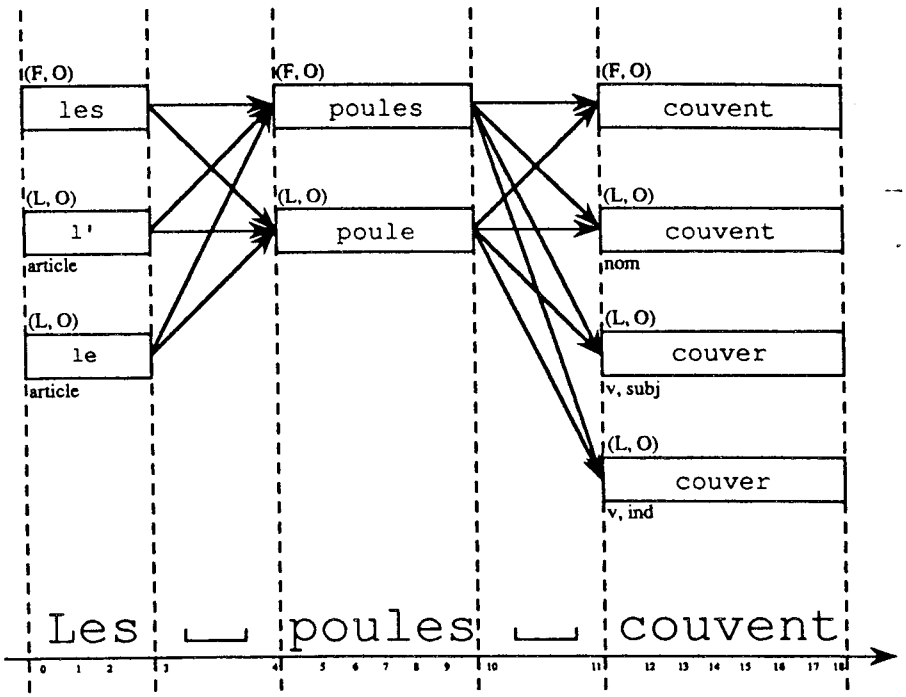
(49) (") (0) (0) (' ') (F) (255)

(50) (") (0) (0) (' ') (F) (255)

Annexe 2 - Exemple de fichier texte résultat d'un lemmatiseur intégrable à ODILE.

L'analyse de la chaîne "les poules couvent" peut donner le fichier texte résultat suivant où les informations optionnelles `Categorie` et `Variables` ne figurent pas.

```
(((Numéro 1) (ValChaîne 'les) (TypeChaîne 'F) (Nature 'O)
(Origine 1) (Longueur 3)) (Suivants (4 5)))

(((Numéro 2) (ValChaîne 'l') (TypeChaîne 'L) (Nature 'O)
(Origine 1) (Longueur 3)) (Suivants (4 5)))

(((Numéro 3) (ValChaîne 'le) (TypeChaîne 'L) (Nature 'O)
(Origine 1) (Longueur 3)) (Suivants (4 5)))

(((Numéro 4) (ValChaîne 'poules) (TypeChaîne 'F) (Nature
'O) (Origine 5) (Longueur 6)) (Suivants (6 7 8 9)))

(((Numéro 5) (ValChaîne 'poule) (TypeChaîne 'L) (Nature 'O)
(Origine 5) (Longueur 6)) (Suivants (6 7 8 9)))

(((Numéro 6) (ValChaîne 'couvent) (TypeChaîne 'F) (Nature
'O) (Origine 11) (Longueur 7)) (Suivants ()))

(((Numéro 7) (ValChaîne 'couvent) (TypeChaîne 'L) (Nature
'O) (Origine 11) (Longueur 7)) (Suivants ()))

(((Numéro 8) (ValChaîne 'couver) (TypeChaîne 'L) (Nature
'O) (Origine 11) (Longueur 7)) (Suivants ()))

(((Numéro 9) (ValChaîne 'couver) (TypeChaîne 'L) (Nature
'O) (Origine 11) (Longueur 7)) (Suivants ()))
```



BIBLIOGRAPHIE

[CHAP89a] L. CHAPON, M. DJAMEI, P. DJOHARIAN, F. MORENO. *Etude et révision du système PILAF. Portage vers le Macintosh. Réalisation d'un lemmatiseur. Description de l'existant.*
Projet DESS-IDC. Mai 1989.

[CHAP89b] L. CHAPON, M. DJAMEI, P. DJOHARIAN, F. MORENO. *Etude et révision du système PILAF. Portage vers le Macintosh. Réalisation d'un lemmatiseur. Rapport final de projet.*
Projet DESS-IDC. Juin 1989.

[CHAU72] J. CHAUCHE, P. GUILLAUME, M. QUEZEL-AMBRUNAZ. *Le sytème ATEF (Analyse de Textes en Etats Finis).*
Document interne GETA. Octobre 1972.

[COUL86] D. COULON, D. KAYSER. *Informatique et langage naturel. Présentation générale des méthodes d'interprétation des textes écrits.*
Technique et sciences informatiques. Vol.5, Numéro 2. 1986.

[COUR77] J. COURTIN. *Algorithmes pour le traitement interactif de langues naturelles.*
Thèse de doctorat d'état (USMG-INPG). Octobre 1977.

[COUR88] J. COURTIN, D. DUJARDIN, D. GENTHIAL, I. KOWARSKI, B. COHARD, V. STRUBE DE LIMA. *Le système PILAF.*
Document interne LGI (IMAG) présenté au PRC Communication Homme-Machine. Février 1988.

[COUT88] J. COUTAZ. *Interface homme-ordinateur: conception et réalisation.*
Thèse de doctorat d'état (UJF). Décembre 1988.

[GRAN75] E. GRANDJEAN. *Conception et réalisation d'un dictionnaire pour un analyseur interactif de langues naturelles.*
Mémoire CNAM. Février 1975.

[KNUT73] D. E. KNUTH. *The art of computer programming, Vol. 3, Sorting and Searching.*
Addison-Wesley. 1973.

[NIGAY90] L. NIGAY. *Modélisation des architectures logicielles des sytèmes interactifs.*
Rapport de DEA d'informatique. Laboratoire de Génie informatique (INPG). Juin 1990.

[PALM90] P. PALMER. *Etude d'un analyseur de surface de la langue naturelle. Application à l'indexation automatique de textes.*
Thèse de doctorat spécialité informatique (UJF). Septembre 1990.

[WIRT76] N. WIRTH. *Algorithms + Data structures = Programs.*
Prentice-Hall. 1976.

REFERENCES TECHNIQUES.

[THIN90] P. BORENSTEIN. *THINK Pascal™. User's manual.*
Symantec. 1990.

[INSM87] INSIDE MACINTOSH™. *Volumes I, II, III.*
Addison-Wesley Publishing Company, Inc. 1987.

[FONT88] P. FONTANET. *Mac School 1. Support de cours.*
Document interne WinSoft. 1988.

[RESM88] MACINTOSH® RESEDIT REFERENCE. *Beta Version. Apple® Technical Publications.*
Apple Computer. 1988.

[TURB86] TURBO PASCAL POUR LE MAC. *Guide de l'utilisateur et manuel de références.Version éducation.*
Borland. 1986.

[WINT87] Manuel d'utilisation de WinTool™. Version 1.1.
WinSoft. 1987.

[WINF90] Manuel d'utilisation de WinFile™. Version 1.1. WinSoft. 1990.

Isabelle TOMASINO

ODILE : un Outil d'Intégration Extensible de Dictionnaires et de Lemmatiseurs.

Mémoire d'ingénieur C.N.A.M. Grenoble 1990.

Ce mémoire traite de la conception et de la réalisation d'un logiciel interactif d'assistance linguistique sur Macintosh. Il doit permettre la mise en relation de textes en langue naturelle avec des dictionnaires usuels.

Pour cela, le logiciel à concevoir doit intégrer et faire coopérer des outils préexistants, à savoir un lemmatiseur, chargé de la fabrication d'entrées potentielles de dictionnaire après analyse du texte en entrée, et un outil d'accès à des dictionnaires. Il doit être le plus universel possible pour pouvoir exploiter la diversité des lemmatiseurs et des outils dictionnaires existants. Les lemmatiseurs PILAF et ATEF et différents outils d'accès aux dictionnaires tels que WinTool™, WinFile™ et Word Finder® sont présentés, avant de fixer notre choix sur PILAF et WinTool™ pour la réalisation de notre prototype. Dans ce mémoire, nous proposons, pour l'intégrateur ODILE, une interface utilisateur qui réponde à un certain nombre de principes ergonomiques. Nous définissons également des interfaces programme générales pour la communication des deux logiciels avec notre outil.

Nous nous sommes également préoccupé de l'évolution de notre outil ; en particulier nous souhaitions en faciliter la maintenance. Le souhait d'une architecture modulaire pour ODILE a été l'occasion, pour nous, d'étudier le modèle PAC de structuration logicielle des systèmes interactifs. Dans l'implémentation actuelle, nous en avons retenu certains aspects intéressants .

Le besoin d'améliorer les performances du lemmatiseur intégré nous a conduit à réviser la structure de son dictionnaire. Nous examinons plusieurs organisations possibles.

mots-clés : traitement de la langue naturelle, communication homme-machine, bureautique, structuration des systèmes interactifs, génie logiciel, lemmatiseur, outil dictionnaire.

keywords : natural language processing, man-machine communication, office automation, interactive system structuration, software engineering, lemmatizer, dictionary tool.