



HAL
open science

Des systèmes de TA homogènes aux systèmes de TAO hétérogènes

Hong-Thai Nguyen

► **To cite this version:**

Hong-Thai Nguyen. Des systèmes de TA homogènes aux systèmes de TAO hétérogènes. Interface homme-machine [cs.HC]. Université Joseph-Fourier - Grenoble I, 2009. Français. NNT: . tel-00447571

HAL Id: tel-00447571

<https://theses.hal.science/tel-00447571>

Submitted on 15 Jan 2010

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.

Université Joseph Fourier de Grenoble

N° attribué par la bibliothèque

/ / / / / / / / / / /

THESE

pour obtenir le grade de

DOCTEUR DE L'UJF

Spécialité : "INFORMATIQUE : SYSTEMES D'INFORMATION"

préparée au laboratoire GETALP-LIG (CNRS-INPG-UJF-UPMF)

dans le cadre de l'Ecole Doctorale "Mathématiques, Sciences et Technologie de l'Information"

présentée et soutenue publiquement

par

M. Hong-Thai NGUYEN

le 18 décembre 2009

**Des systèmes de TA homogènes
aux systèmes de TAO hétérogènes**

JURY

M. Laurent BESACIER	Président
M. Jacques CHAUCHÉ	Rapporteur
M. Denis MAUREL	Rapporteur
M. Jesús CARDEÑOSA	Rapporteur
M. Vincent BERMENT	Examineur
M. Mathieu LAFOURCADE	Examineur
M. Christian BOITET	Directeur de thèse
M. Éric CASTELLI	Co-directeur de thèse

À notre ange

Remerciements

Mes remerciements les plus vif vont d'abord à mon directeur de thèse, Christian Boitet, qui m'a non seulement encadré en bonne voie scientifique dans cette thèse, mais encore m'a passé l'enthousiasme et l'esprit de recherche lors des conseils et des supports prestigieux. Sans lui, je ne serais jamais arrivé à découvrir un domaine si intéressant et prometteur de Traduction Automatique.

Je tiens à remercier à Éric Castelli, qui m'a accueilli au MICA, Vietnam dans le cadre de cette thèse cotutelle.

Je remercie Jacques Chauché, Denis Maurel et Jesús Cardenosa d'avoir accepté d'être mes rapporteurs de thèse, ainsi que Vincent Berment et Mathieu Lafourcade d'avoir accepté d'être mes examinateurs, et Laurent Besacier, qui a bien voulu présider le jury.

J'adresse également mes remerciements à l'ambassade de France au Vietnam, dont la bourse m'a permis de mener cette thèse à bien. Je n'oublie non plus pas mon long stage à la société Systran, lors duquel j'ai pu travailler directement sur une famille de systèmes de TA commercialisés.

Je voudrais aussi remercier les membres du GETALP (ex GETA), qui m'ont offert une ambiance favorable pour réaliser ce travail. J'aimerais mentionner particulièrement : Jean-Claude et Jean-Phillipe pour leurs conseils et explications sur le système Ariane-G5 et le projet Ariane-Y, Valérie pour des discussions intéressantes dans le cadre du projet iMAG, et en particulier ses contributions et celles d'Hervé lors de mes répétitions, Gilles et Mathieu M. pour leurs conseils techniques et leurs critiques très constructives, et enfin Didier et David pour notre coopération efficace et sympathique dans le cadre du projet OMNIA.

Je tiens également à remercier Étienne, Damien et Virginie pour toute la partie concernant le méta-EDL CASH et le méta-EDL générique WICALE. Sans eux, cette thèse n'aurait jamais été commencée.

Un grand merci à mes amis Vietnamiens à Grenoble, en particulier « anh Phap » pour son travail énorme sur la gestion de corpus et dans le projet iMAG, ainsi qu'aux amis avec lesquels j'ai partagé des moments inoubliables, en particulier Lexu, Trang S&L, PhAnh, Trung beo... J'adresse également aux amis Sigrig & Stéfan et Émile. Grâce à eux, je trouve que Grenoble est une ville magnifique.

Enfin, je voudrais exprimer mes sentiments les plus affectueux pour Trang, ma chère épouse, et pour mes parents qui ont supporté mes longues absences et m'ont énormément soutenu tout au long de mes séjours en France.

TABLE DES MATIERES

INTRODUCTION.....	1
PARTIE 1. VERS DES SYSTÈMES DE TRADUCTION ASSISTÉE PAR L'ORDINATEUR HÉTÉROGÈNES.....	3
INTRODUCTION.....	3
CHAPITRE 1. TA ET THAM.....	4
INTRODUCTION.....	4
1.1. AJOUT DE FONCTIONS DE THAM À LA TA.....	4
1.1.1. <i>Mode de révision en Ariane-G5</i>	4
1.1.2. <i>PAHO</i>	5
1.1.3. <i>Pensée, Yakushite.net</i>	6
1.1.4. <i>BehaviorTran de Keh-Yih Su (Taipeh), amélioration de la TA par le retour des traducteurs</i>	7
1.2. AJOUT DE COMPOSANTS DE TA À LA THAM.....	7
1.2.1. <i>Intégration d'aides dictionnairiques pour la THAM</i>	7
1.2.1.1. Aide pure (dico « main gauche » d'Alain K.Melby).....	7
1.2.1.2. Système de THAM Transactive (ALPS).....	7
1.2.1.3. Système de THAM de Weidner.....	8
1.2.2. <i>Ajout de mémoires de traductions</i>	8
1.2.2.1. Systèmes usuels de gestion de MT.....	9
1.2.2.1.1. TM/2 (IBM).....	9
1.2.2.1.2. Trados (SDL).....	9
1.2.2.1.3. DéjàVu (ATRIL).....	10
1.2.2.1.4. Transit (STAR).....	11
1.2.2.2. XTM (XML-INTL), système de MT sur le Web.....	12
1.2.2.3. Similis (Lingua&Machina), nouvelle génération de MT.....	12
1.2.3. <i>Ajout ou intégration de TA à des systèmes de THAM</i>	13
1.2.3.1. Appel de systèmes de TA « experts ».....	13
1.2.3.1.1. Intégration de LMT à TM/2 (IBM).....	13
1.2.3.1.2. Projet Eurolang Optimizer avec appel de LOGOS, METAL et Ariane-G5.....	13
1.2.3.2. Perspectives: création d'un système de TA à partir des ressources de THAM.....	14
1.2.3.2.1. Méthodes empiriques pour la création de systèmes de TA.....	14
1.2.3.2.1.1. TA probabiliste.....	14
1.2.3.2.1.2. TA par l'exemple.....	15
1.2.3.2.2. Avancées dans la TA « experte » avec dictionnaires et corpus communs.....	15
1.2.3.2.3. Création d'un système de TA « expert » à partir des ressources d'un système à MT.....	16
1.2.3.2.3.1. Similis, transformation de THAM en TA par l'exemple.....	16
1.2.3.2.3.2. Construction d'un système de TA à partir d'amphigrammes.....	17
CONCLUSION.....	18
CHAPITRE 2. ÉVOLUTION VERS LES SYSTÈMES DE TA HÉTÉROGÈNES.....	19
INTRODUCTION.....	19
2.1. SYSTÈMES DE TA « MULTIPLES ».....	19
2.1.1. <i>Exemples de systèmes connus</i>	19
2.1.1.1. Pangloss (1995-1996), un système multiple avec même support et format informatique.....	19
2.1.1.2. VERBMOBIL (2000), des modules différents mais tous sous Linux.....	20
2.1.1.3. ALTFLASH (2001), un système de « TA par patrons » avec un système général en secours.....	20
2.1.2. <i>Idées en cours, combinaison de résultats de plusieurs systèmes de TA</i>	21
2.1.2.1. Récupération de résultats de systèmes de TA en ligne.....	21
2.1.2.1.1. Soumission et récupération de plusieurs systèmes de TA Web.....	21
2.1.2.1.2. Utilisation de résultats multiples factorisés.....	21
2.1.2.1.3. Mise en service en ligne de systèmes de TA locaux.....	22
2.1.2.2. Architecture opérationnelle pour les systèmes de TA multiples dans le Web 2.0.....	23
2.1.2.2.1. Problème d'organisation dans l'utilisation des contributions publiques de Google Translate.....	23
2.1.2.2.2. iMAG, passerelles d'accès multilingue dédiées.....	23
2.2. SYSTÈMES DE TA AVEC PROCESSUS HÉTÉROGÈNES.....	25
2.2.1. <i>Avènement de la TA « fondée sur la connaissance » (1985-1995)</i>	25
2.2.1.1. Interaction avec une base de connaissances (thèse Gerber 1984).....	25
2.2.1.2. Interaction avec une ontologie dans KBMT-89, puis système KANT-Catalyst.....	26
2.2.1.3. Interaction entre la connaissance et le modèle statistique pour traduction de parole, système MASTOR (IBM).....	28
2.2.2. <i>Interaction avec des humains</i>	29

Table des matières

2.2.2.1.	Interaction en analyse et transfert à ITS-1 (Eldon G.Lytle, BYU, Provo, 1972-1981).....	29
2.2.2.2.	Interaction le plus tard possible dans ITS-2 (Éric Wehrli, Genève, LATL).....	29
2.2.2.3.	Interaction entre 2 phases dans le projet LIDIA (1989-1996).....	30
2.2.3.	<i>Architectures computationnelles et EDL différents</i>	31
2.2.3.1.	Maquette MU → FR (1984-1985).....	32
2.2.3.2.	Le projet MMT du CICC (1987-1993).....	32
2.2.3.3.	Le projet UNL (1996-).....	33
2.3.	VERS DES SYSTÈMES DE TA ET DE THAM INTÉGRÉS & COLLABORATIFS.....	35
2.3.1.	<i>Systèmes de TH collaboratifs</i>	35
2.3.1.1.	Localisation interne de logiciels libres.....	36
2.3.1.1.1.	MLP, un projet de localisation de logiciels.....	36
2.3.1.1.2.	Launchpad Translation, un outil de gestion de projets de localisation logicielle.....	36
2.3.1.1.3.	Traduction de Facebook par ses utilisateurs.....	36
	Caractéristiques communes.....	37
2.3.1.2.	Wikisation, communautés de contributeurs bénévoles.....	37
2.3.1.2.1.	TraduWiki.....	37
2.3.1.2.2.	Pootle, un portail de gestion de traductions en ligne.....	38
	Caractéristiques communes.....	38
2.3.1.3.	Support linguistique pour des projets de traduction pour des « causes ».....	38
2.3.1.3.1.	QRLex.....	38
2.3.1.3.2.	Projet Pax Humana, Amnesty International.....	38
2.3.1.3.3.	Récapitulatif.....	38
2.3.2.	<i>Systèmes de TH intégrant l'appel à des systèmes de TA</i>	39
2.3.2.1.	TransActive (ALPS, 1983-1990).....	39
2.3.2.2.	Eurolang Optimizer (1992-1996).....	39
2.3.2.3.	Prototype BEYTrans.....	39
2.3.2.4.	SECTra_w 2.0 dans le sous-projet EOLSS/UNL-FR de post-édition de TA.....	39
	Récapitulatif.....	40
2.3.3.	<i>Systèmes de TA intégrant des fonctions de TH</i>	40
2.3.3.1.	Avant le Web 2.0.....	40
2.3.3.1.1.	Ariane-78 (1978-1985), puis Ariane-G5 (1985-).....	40
2.3.3.1.2.	Systèmes de TA japonais (1982-).....	41
2.3.3.1.3.	SPANAM & ENGSPAN (1984-).....	41
2.3.3.1.4.	LOGOS avec Wang.....	41
2.3.3.2.	Avec le Web 2.0.....	41
2.3.3.2.1.	Yakushite.net: intégration de contributions lexicales au système Pensée.....	41
2.3.3.2.2.	SYSTRANet.....	41
2.3.3.2.3.	Google Translator Toolkit pour traduire des documents en ligne.....	42
	CONCLUSION.....	42
	CHAPITRE 3. UN EXEMPLE COMPLET : ARIANE-G5 ET SON EDL	44
	INTRODUCTION.....	44
3.1.	PRINCIPES GÉNÉRAUX.....	45
3.1.1.	<i>Environnement matériel et logiciel</i>	45
3.1.1.1.	Système d'exploitation et plates-formes.....	45
3.1.1.2.	Instances installées.....	46
3.1.1.3.	Environnement actuel.....	47
3.1.2.	<i>Organisation logique</i>	47
3.1.2.1.	Étapes, phases et articulations du processus de traduction.....	47
3.1.2.1.1.	Ariane-78.....	47
3.1.2.1.2.	Ariane-G5.....	48
3.1.2.2.	Types de données essentiels et terminologie de base.....	49
3.1.2.3.	Composants et variantes d'une phase.....	51
3.2.	COMPOSANTS LOGICIELS.....	52
3.2.1.	<i>Moniteur interactif natif</i>	52
3.2.2.	<i>Le réseau LIDIA</i>	53
3.2.2.1.	Composants logiques.....	54
3.2.2.2.	Organisation des machines virtuelles sur l'hôte IBM sous VM puis zVM.....	54
3.2.2.3.	Implémentation.....	56
3.2.3.	<i>LSPL</i>	56
3.2.3.1.	Nature, raisons de leur variété.....	56
3.2.3.1.1.	Nature.....	56
3.2.3.1.2.	Raisons.....	56
3.2.3.2.	Les LSPL d'Ariane-G5.....	57
3.2.3.2.1.	AM & GM.....	57
3.2.3.2.1.1.	ATEF, un langage pour l'analyse morphologique.....	57

3.2.3.2.1.2.	SYGMOR, un langage pour la génération morphologique	59
3.2.3.2.2.	Phase de transformation d'arbres décorés	60
3.2.3.2.2.1.	TRACOMPL, pour la transformation de décorations (« articulations »)	60
3.2.3.2.2.2.	ROBRA, pour l'écriture de systèmes transformationnels d'arbres décorés	60
3.2.3.2.2.3.	EXPANS/TRANSF, pour l'expansion lexicale monolingue & le transfert lexical bilingue	61
3.2.3.3.	LSPL associés à Ariane-G5	62
3.2.3.3.1.	ATLAS, pour l'aide à l'indexage dans les dictionnaires des LSPL	62
3.2.3.3.2.	LT, un langage d'écriture de transpositeurs	62
3.2.3.3.3.	TTEDIT, un éditeur transformationnel d'arbres décorés	63
3.3.	COMPOSANTS LOGICIELS	63
3.3.1.	Organisation des dictionnaires	63
3.3.1.1.	Notions utilisées	63
3.3.1.2.	Espaces lexicaux	64
3.3.1.2.1.	Organisation	64
3.3.1.2.2.	Traitement des UL dynamiques	65
3.3.1.2.3.	Représentations usuelles des « acceptions » (ou « sens de mots »)	65
3.3.1.3.	Fonctions utilitaires	65
3.3.1.3.1.	Indexage	65
3.3.1.3.2.	Tris & listes	65
3.3.1.3.3.	Vérification individuelle et vérification « croisée »	66
3.3.2.	Traitement des corpus	66
3.3.2.1.	Conventions de nommage	66
3.3.2.2.	Organisation des fichiers pour une traduction	66
3.3.2.3.	Types de fichiers de résultat	67
3.3.2.3.1.	Résultat intermédiaire (liste d'arbres décorés)	67
3.3.2.3.2.	Traduction brute	67
3.3.2.3.3.	Révision	67
3.3.3.	Programmation linguistique	68
3.3.3.1.	Méthodologie directe	68
3.3.3.2.	Grammaires statiques pour la spécification semi-formelle	68
3.3.3.3.	Stratégie d'utilisation des LSPL	68
3.3.3.3.1.	ATEF	68
3.3.3.3.2.	ROBRA	69
3.4.	LE PROJET ARIANE-Y ET L'INTÉGRATION DE LSPL VARIÉS (ARIANE-Y COMPLET)	70
3.4.1.	Limites d'Ariane-G5 à lever	70
3.4.1.1.	Limites d'implémentation	70
3.4.1.1.1.	Compilation non incrémentale	70
3.4.1.1.2.	Accessibilité par réseau mais non-portabilité	70
3.4.1.1.3.	Diversité des langages de programmation	71
3.4.1.1.4.	Architecture du processus de traduction pas assez générique	71
3.4.1.2.	Limitation à certains LSPL	71
3.4.1.2.1.	Désir naturel d'intégrer d'autres LSPL	71
3.4.1.2.2.	Absence de phases à architecture computationnelle empirique	71
3.4.1.3.	Mise en réseau complexe	71
3.4.1.3.1.	Qualité de l'idée de base	71
3.4.1.3.2.	Implémentation complexe car en avance sur le Web	72
3.4.1.3.3.	Esquisse d'une solution : des machines virtuelles vers des serveurs et/ou agents	72
3.4.2.	Grandes stratégies retenues	72
3.4.2.1.	Simplifier pour généraliser	72
3.4.2.2.	Portabilité et extensibilité	72
3.4.2.3.	Ouverture à la TAO hétérogène	73
3.4.3.	Architecture générale	73
CONCLUSION		73
PARTIE 2. MÉTHODES POUR LA CONVERGENCE TA ↔ THAM		75
INTRODUCTION		75
CHAPITRE 4. INTERVENTION DE NON-SPÉCIALISTES SUR LES COMPOSANTS LOGICIELS DE LA TA		76
INTRODUCTION		76
4.1. ASPECTS LIÉS AUX EDL		76
4.1.1. Délocalisation par création d'un méta-EDL		77
4.1.1.1. Technique externe de mise en réseau d'Ariane-G5 (réseau LIDIA)		77
4.1.1.2. CASH (CASHrev) pour Ariane-G5		77
4.1.2. TA hétérogène par méta-EDL générique (WICALE)		78
4.1.2.1. Métalangage pour définir les formats d'échange et d'interface		78

Table des matières

4.1.2.1.1.	Motivations et objectifs	78
4.1.2.1.2.	Exemple pour Ariane-G5.....	79
4.1.2.1.2.1.	Architecture de données.....	79
4.1.2.1.2.2.	Commandes.....	80
4.1.2.1.3.	Exemple pour PILAF.....	82
4.1.2.1.3.1.	Architecture de données.....	83
4.1.2.1.3.2.	Commandes.....	84
4.1.2.2.	Exemple d'intégration de fonctionnalités utiles	86
4.1.2.2.1.	Édition « ouverte »	86
4.1.2.2.1.1.	Techniques	86
4.1.2.2.1.2.	Solutions.....	86
4.1.2.2.1.3.	Démonstration.....	87
4.1.2.2.2.	Navigation « à la Doxygen ».....	88
4.1.2.2.3.	Extensions possibles	90
4.1.2.2.3.1.	Édition « directe » d'objets linguistiques.....	90
4.1.2.2.3.2.	Reimplémentation progressive de LSPL dans WICALE.....	90
4.1.3.	<i>Application au moniteur EMEU_w</i>	90
4.1.3.1.	Contexte et objectifs d'EMEU_w.....	90
4.1.3.2.	Technique inspirée de WICALE.....	92
4.1.3.3.	Validation et exemple de connexion avec SECTra_w	92
4.2.	RÉINGÉNIÉRIE DES LSPL EN ARIANE-Y	93
4.2.1.	<i>Principes et contraintes du projet Ariane-Y</i>	93
4.2.1.1.	Innovations logicielles pour la TA.....	94
4.2.1.1.1.	Principe de permanence des objets en mémoire et modification incrémentale.....	94
4.2.1.1.2.	Pas de limitation de taille.....	94
4.2.1.1.3.	Compilation en ANTLR et gestion des erreurs « sémantiques » au chargement.....	94
4.2.1.2.	Organisation de la structure de travail.....	96
4.2.1.2.1.	SI (Structure Interne) = SA (Structure Abstraite) + EPA (Extensions pour Présentations & Algorithmes).....	96
4.2.1.2.2.	Rôle de XML.....	97
4.2.1.2.3.	REXX et REXXstack.....	97
4.2.1.3.	Principe d'architecture.....	100
4.2.1.3.1.	Délégation.....	100
4.2.1.3.2.	Agents à gros grains.....	100
4.2.1.3.3.	Possibilité de plusieurs moniteurs en parallèle.....	100
4.2.2.	<i>Implémentation des LSPL : exemple des systèmes-Q</i>	100
4.2.2.1.	Systèmes-Q.....	101
4.2.2.1.1.	Rappel de la syntaxe et la sémantique opérationnelle des systèmes-Q	101
4.2.2.1.1.1.	Syntaxe.....	101
4.2.2.1.1.2.	Sémantique opérationnelle.....	102
4.2.2.1.1.3.	Exemples de règles utiles.....	104
4.2.2.1.2.	Reconstitution d'un algorithme adapté.....	104
4.2.2.1.2.1.	Algorithme reconstitué.....	104
4.2.2.1.2.2.	Structure de données.....	107
4.2.2.2.	Points intéressants de l'implémentation reliés aux principes concernés.....	109
4.2.2.2.1.	Compilation en ANTLR et chargement d'une SI	109
4.2.2.2.2.	Moteur.....	110
4.2.2.2.3.	Moniteur simple utilisant REXX	110
4.2.2.3.	Évaluation & perspectives.....	111
4.2.2.3.1.	Test et mesures sur des exemples connus.....	111
4.2.2.3.2.	Exemple de l'annotation de textes dans le projet OMNIA.....	112
4.2.2.3.3.	Variantes possibles.....	113
4.2.2.3.3.1.	Étiquettes étendues aux caractères Unicode	113
4.2.2.3.3.2.	Variantes du moteur pour le « mode dictionnaire »	114
4.2.2.3.3.3.	Compilation incrémentale/efficacité	114
4.3.	ASPECTS LIÉS AU MONITEUR	114
4.3.1.	<i>Idée d'un moniteur ouvert fondé sur un langage de commandes</i>	114
4.3.2.	<i>Réalisations possibles en utilisant REXX et les REXXstacks</i>	115
	CONCLUSION	116
	CHAPITRE 5. INTÉGRATION DE COMPOSANTS LINGUICIELS POUR LA TA ET POUR LA THAM	117
	INTRODUCTION.....	117
5.1.	BASE LEXICALE, INTÉGRATION DE PIVAX À ARIANE-Y	117
5.1.1.	<i>Importance primordiale des connaissances lexicales en TA</i>	117
5.1.1.1.	Nécessité pour toutes les techniques de TA.....	117

5.1.1.2.	Application à tous les niveaux lexicaux (mots, idiomes, fragments phraséologiques...)	117
5.1.1.3.	Partie majeure du coût de développement	117
5.1.2.	<i>Solution pour Ariane-Y : délégation + idée de « serveur lexical » construit comme un agent autonome</i>	118
5.1.3.	<i>Nouveaux problèmes et solutions envisagées</i>	118
5.1.3.1.	Problème de délais	118
5.1.3.2.	Utilisation de « boîtes aux lettres » avec calcul incrémental et boucles infinies	118
5.2.	CORPUS : INTÉGRATION DE SECTRA_w À ARIANE-Y	119
5.2.1.	<i>Corpus brut, corpus révisé de TA</i>	119
5.2.2.	<i>Solution pour Ariane-Y : délégation + idée de « serveur corporal »</i>	119
5.2.3.	<i>Nouveaux problèmes</i>	120
5.2.3.1.	Hétérogénéité comme pour la base lexicale	120
5.2.3.2.	Difficulté de concevoir une structure générique	121
5.2.3.3.	Passage à l'échelle	121
5.3.	DIFFICULTÉS CONCEPTUELLES ET PRATIQUES AU NIVEAU DES GRAMMAIRES ET DES AUTOMATES	122
5.3.1.	<i>Niveaux conceptuels</i>	122
5.3.1.1.	Particularité dans les grammaires	122
5.3.1.2.	Différents algorithmes pour un même modèle	122
5.3.1.3.	Transformation entre les grammaires	122
5.3.1.3.1.	Possibilité pour les grammaires « dynamiques »	123
5.3.1.3.2.	Possibilité pour les grammaires statiques	123
5.3.1.3.2.1.	Spécification semi-formelle	123
5.3.1.3.2.2.	Spécification formelle	124
5.3.1.3.2.3.	Conclusion sur les informations partageables	124
5.3.2.	<i>Niveaux pratiques</i>	124
5.3.3.	<i>Conclusion pour la piste d'une base grammaticale commune</i>	124
	CONCLUSION	124
CHAPITRE 6. INTÉGRATION D'OUTILS ET TECHNIQUES PROVENANT DE LA THAM		126
	INTRODUCTION	126
6.1.	PROBLÈMES ET SOLUTIONS POUR INTÉGRER L'USAGE DE MÉMOIRES DE TRADUCTION À UN SYSTÈME DE TA	126
6.1.1.	<i>Unification entre résultats de TA et de recherche dans une MT intégrée</i>	126
6.1.1.1.	Méthodes de recherche dans les MT	126
6.1.1.1.1.	Méthode exacte	126
6.1.1.1.2.	Recherche approchée	126
6.1.1.2.	Problème d'unification des résultats et prise en compte des retours des traducteurs	127
6.1.1.3.	Problèmes de présentation	127
6.1.2.	<i>Spécialisation de systèmes de TA généraux</i>	127
6.1.2.1.	Adaptation/Spécification de systèmes « experts »	127
6.1.2.2.	Adaptation en TA probabiliste	128
6.1.2.2.1.	Limites	128
6.1.2.2.2.	Solutions retenues	128
6.1.2.2.2.1.	Annotation, outil pour valider les annotations associées	128
6.1.2.2.2.2.	Renvoyer des retours, suivre la méthode de Su Keh Yi	128
6.1.2.3.	Adaptation en TA par l'exemple	128
6.1.2.3.1.	TA par analogie de chaînes	128
6.1.2.3.2.	TA par alignements structurels	128
6.1.3.	<i>Création de systèmes de TA spécialisés à partir d'une MT</i>	129
6.2.	UTILISATION DE DICTIONNAIRES DE THAM DANS DES SYSTÈMES DE TA	129
6.2.1.	<i>Problèmes d'intégration des dictionnaires de THAM et de TA</i>	129
6.2.1.1.	Homogénéité de format	129
6.2.1.2.	Modification	130
6.2.1.3.	Compétences des utilisateurs	130
6.2.1.4.	Opérations	130
6.2.2.	<i>Solution retenue de la base lexicale multilingue unique</i>	130
6.2.3.	<i>Principes de réalisation</i>	130
6.2.3.1.	Construction d'une base lexicale pour la TA et la THAM	130
6.2.3.2.	Intégration aux systèmes de THAM	130
6.2.3.3.	Offre d'outils de développements lexicaux intuitifs	131
6.3.	UNIFICATION DES INTERFACES DE TH, DE POST-ÉDITION DE TA ET D'ÉVALUATION DE TA	131
6.3.1.	<i>Similarité des interfaces</i>	131
6.3.2.	<i>Unification déjà en cours</i>	131

Table des matières

6.3.3.	<i>Technique de réalisation à l'époque du Web 2.0</i>	131
6.3.3.1.	Web ou pas Web.....	131
6.3.3.2.	Utilisation d'un navigateur standard comme base de l'IHM.....	132
6.3.3.3.	IHM dépouillée (« bare bones ») et fonctionnelle à la Excel et pas d'intégration WYSIWYG 132	
	CONCLUSION.....	132
	PARTIE 3. GESTION DES BASES DE CONNAISSANCES LEXICALES ET INTÉGRATION.....	133
	INTRODUCTION.....	133
	CHAPITRE 7. PIVAX, UNE BASE LEXICALE POUR LES SYSTÈMES DE TA HÉTÉROGÈNES.....	135
	INTRODUCTION.....	135
7.1.	GESTION DES CONNAISSANCES LEXICALES EN TA HOMOGÈNE.....	136
7.1.1.	<i>Fichiers et BDLex « implicite » à la VISULEX</i>	136
7.1.2.	<i>Vraie BDLex sans possibilité d'interaction directe avec les systèmes de TA</i>	138
7.1.2.1.	BDAO.....	139
7.1.2.2.	UWGate.....	139
7.1.2.3.	PARAX.....	139
7.1.3.	<i>BDLM avec interaction limitée à la consultation durant la TA</i>	140
7.1.3.1.	MU (Adabase).....	140
7.1.3.2.	METAL.....	141
7.1.3.3.	LMT.....	141
7.1.4.	<i>BDLM avec consultation et édition durant la TA : Pensée et yakushite.net (Oki)</i>	141
7.2.	PROBLÈMES POSÉS PAR L'ÉLARGISSEMENT À UN ENSEMBLE DE SYSTÈMES ARBITRAIRES DE TA 142	
7.2.1.	<i>Variété dans la nature des données</i>	142
7.2.1.1.	Information.....	142
7.2.1.2.	Structure.....	142
7.2.2.	<i>Variété des opérations sur l'information lexicale</i>	144
7.2.2.1.	Récupération et gestion.....	144
7.2.2.2.	Développement lexical.....	144
7.2.2.3.	Programmabilité.....	144
7.2.3.	<i>Problèmes pratiques</i>	144
7.2.3.1.	Partage et protection des données.....	145
7.2.3.2.	Synchronisation.....	145
7.2.3.3.	Synthèse des problèmes.....	145
7.3.	VERS UNE SOLUTION LIMITÉE AUX SYSTÈMES DE TAO À « PIVOT LEXICAL ».....	145
7.3.1.	<i>Travaux précédents de construction d'une BDLM pour la TA ou la TH</i>	145
7.3.1.1.	Solution « DGT-86 » : N volumes multicibles (« furcoïdes »).....	145
7.3.1.2.	Projet Papillon, une BDLM pour plusieurs dictionnaires.....	146
7.3.1.2.1.	Papillon NADIA.....	146
7.3.1.2.2.	Papillon CDM.....	147
7.3.1.3.	Plate-forme Jibiki.....	149
7.3.2.	<i>Principes d'une solution limitée aux systèmes partageant le pivot lexical</i>	149
7.3.2.1.	Macrostructure à avec trois niveaux (lexie, axème, axie).....	149
7.3.2.2.	Microstructure avec division des données en public/privé.....	151
7.3.2.3.	Implémentation.....	152
7.3.3.	<i>Expérimentation sur trois projets</i>	152
7.3.3.1.	UNL/U++C.....	152
7.3.3.1.1.	Le problème d'unification des ressources lexicales (UW).....	153
7.3.3.1.2.	Modélisation et montage de la base initiale.....	154
7.3.3.1.2.1.	Définition des microstructures et de la macrostructure.....	154
7.3.3.1.2.2.	Gestion des dictionnaires UW-LN.....	154
7.3.3.1.2.3.	Import de ressources lexicales externes (TER de G. Ginon).....	155
7.3.3.1.3.	Solutions possibles pour unifier les UW.....	155
7.3.3.2.	Gestion de dictionnaires de SYSTRAN.....	156
7.3.3.3.	Génération de dictionnaires spécifiques dans le projet OMNIA.....	156
	CONCLUSION.....	157
	CHAPITRE 8. ÉVOLUTION DE PIVAX DE LA TA HÉTÉROGÈNE À LA TAO HÉTÉROGÈNE 158	
	INTRODUCTION.....	158
8.1.	PIVAX COMME UNE BDLM CONTRIBUTIVE.....	158
8.1.1.	<i>Expériences sur des environnements de développement lexical</i>	158
8.1.2.	<i>Scénario</i>	159

8.1.3.	<i>Spécification externe</i>	160
8.1.3.1.	Interface de navigation à la PARAX.....	160
8.1.3.2.	Interface d'édition d'une entrée.....	160
8.1.3.3.	Recherche simple et avancée.....	161
8.1.3.4.	Profil d'utilisateur.....	162
8.2.	NOUVEAUX COMPOSANTS POUR L'UTILISATION DIRECTE EN THAM.....	163
8.2.1.	<i>Recherche exacte et approchée intégrant la lemmatisation</i>	163
8.2.2.	<i>Intégration des aides de développement lexical avancées</i>	164
8.2.2.1.	Indexage (ATLAS).....	164
8.2.2.2.	Outil intuitif de manipulation de graphe.....	164
8.2.2.3.	Intégration des outils de gestion de flot de travaux.....	164
8.2.3.	<i>Préremplissage de données lexicales par des agents externes</i>	165
8.2.3.1.	Robodico.....	165
8.2.3.2.	Outils d'extraction de termes bilingues à partir de corpus.....	166
8.3.	INTÉGRATION DE PIVAX DANS DES SYSTÈMES DE THAM.....	167
8.3.1.	<i>Fonctions demandées</i>	167
8.3.1.1.	Dictionnaire minimal pour les données dédiées.....	167
8.3.1.2.	Calcul proactif et synchronisation.....	168
8.3.1.3.	Échange à la WICALE et évolution vers l'architecture en agents à gros gains.....	168
8.3.2.	<i>Application dans des projets réels de traduction</i>	169
8.3.2.1.	EOLSS/UNL-FR.....	169
8.3.2.2.	iMAG.....	170
8.3.3.	<i>Extension de PIVAX pour supporter des bases « préterminologiques »</i>	171
8.3.3.1.	Motivation.....	171
8.3.3.2.	Projets et travaux en cours.....	171
8.3.3.2.1.	Le projet DSR et la thèse de M. Daoud.....	171
8.3.3.2.2.	IToldU, une base développée en contexte d'enseignement.....	171
8.3.3.3.	Réalisation dans PIVAX.....	172
8.3.3.3.1.	Modification effectuée sur PIVAX.....	172
8.3.3.3.2.	Organisation d'une BDLMpT par rapport aux niveaux de confiance.....	173
8.3.3.3.3.	Perspective de la spécification d'interface et support du flot de travaux.....	174
	CONCLUSION.....	174
	CHAPITRE 9. VERS LA PROGRAMMABILITÉ DES BASES LEXICALES PIVAX.....	175
	INTRODUCTION.....	175
9.1.	NIVEAUX DE PROGRAMMATION SOUHAITÉS.....	175
9.1.1.	<i>Support du développement humain</i>	175
9.1.1.1.	Gestion des contributions.....	175
9.1.1.1.1.	Notions de projet, de sous-projet.....	176
9.1.1.1.2.	Suivi des contributions.....	176
9.1.1.1.3.	Flot de travaux.....	176
9.1.1.2.	Langage narratif de programmation pour des tâches simples.....	176
9.1.2.	<i>Programmation possible sur les données lexicales</i>	177
9.1.2.1.	Intégration de bibliothèques adaptées.....	177
9.1.2.2.	Primitives de base comme actions de manipulation directe du réseau lexical.....	177
9.1.2.3.	Langage narratif pour la programmation par des non-informaticiens.....	179
9.1.3.	<i>Programmation de tâches au niveau du système</i>	179
9.1.3.1.	Synchronisation.....	179
9.1.3.2.	Traitements en « boucle infinie » par des agents externes.....	180
9.1.3.3.	Métalangage pour l'échange de données et de commandes pour les agents ou les systèmes lexicaux.....	180
9.2.	UN PREMIER LANGAGE NARRATIF POUR LA MANIPULATION DIRECTE DES GRAPHS LEXICAUX DE PIVAX.....	180
9.2.1.	<i>Spécification</i>	181
9.2.1.1.	Syntaxe.....	181
9.2.1.2.	Exemples de traitement.....	181
9.2.2.	<i>Implémentation</i>	182
9.2.2.1.	Représentation par un graphe MuLLinG.....	182
9.2.2.2.	Algorithme pour charger le graphe de la base de données.....	183
9.2.2.3.	Mise à jour différée de la base.....	184
9.2.3.	<i>Interface de prototype</i>	185
9.3.	PREMIER ALGORITHME GLOBAL POUR UNIFIER LES UW D'UNL.....	185
9.3.1.	<i>Spécification</i>	185
9.3.2.	<i>Réalisation</i>	186
9.3.3.	<i>Résultats</i>	187

CONCLUSION	188
SYNTHESE & PERSPECTIVES.....	189
BIBLIOGRAPHIE.....	191
NETOGRAPHIE.....	206
ANNEXES	209

TABLES DES FIGURES

Figure 1: Un texte ouvert en mode « révision » sous Ariane-G5.....	4
Figure 2: Présentation en colonnes dans Ariane-G5	5
Figure 3: Flot de travaux dans le processus de traduction à PAHO	6
Figure 4: Liaison entre la base lexicale yakushite.net et le système de TA Pensée.....	7
Figure 5: Interface de traduction de TM/2.....	9
Figure 6: Capture d'écran dans le mode de traduction interactive de SDL TRADOS.....	10
Figure 7: Capture d'écran de l'environnement de traduction de DéjàVu.....	11
Figure 8: Interface de traduction de Transit	11
Figure 9: Écran d'interface de post-édition avec recherche de dictionnaire intégrée	12
Figure 10: Exemple de l'extraction de parties de phrase (chunks) par Similis à partir d'un document source et de sa traduction	13
Figure 11: 7 étages de TELA.....	16
Figure 12: Exemple de TransTree.....	17
Figure 13: Architecture du système multi-moteur Pangloss	19
Figure 14: Architecture multi-moteur du système Vermobil	20
Figure 15: Structure de traduction dans ALTFLASH.....	21
Figure 16: Méta-système TRADOH de traduction automatique multilingue	21
Figure 17: Représentation des résultats de TA par un graphe avec manipulation directe pour sélectionner le meilleur.....	22
Figure 18: Architecture composée de SECTra_w, PIVAX, iMAG.....	25
Figure 19: Architecture du prototype de TA experte hétérogène de R.Gerber (1984).....	25
Figure 20: Architecture du système KANT.....	27
Figure 21: Architecture du système KANTOO.....	28
Figure 22: Système de traduction de parole MASTOR d'IBM	28
Figure 23: Construction d'un arbre de question en vue externe de LIDIA.....	30
Figure 24: Architecture logicielle mise en oeuvre en LIDIA-2	31
Figure 25: Maquette de système de TA hétérogène japonais-français combinant MU et Ariane/RUB-FRB [Koca & Sato, 1985].....	32
Figure 26: Architecture du système UNL.....	34
Figure 27: Exemple d'un graphe d'UNL	34
Figure 28: Architecture du déconvertisseur du français du GETALP	35
Figure 29: Interface de l'environnement de traduction de l'interface de Facebook	37
Figure 30: Interface de SECTra_w . 2.0	40
Figure 31: Interface de traduction de Google Translator Toolkit	42
Figure 32: Étapes/Phases dans Ariane-78.....	48
Figure 33: Étapes/Phases dans Ariane-G5.....	49
Figure 34: Composition des logiciels d'Ariane-G5.....	51
Figure 35: Organisation de principe des machines virtuelles constituant le réseau LIDIA	55
Figure 36: Arborecence de choix.....	58
Figure 37: Exemple de la convention de nommage de corpus du système Ariane-G5	66
Figure 38: Format de résultat intermédiaire de traduction	67
Figure 39: Schéma général de l'architecture logicielle d'Ariane-Y	73
Figure 40: Interface de CASH.....	78
Figure 41: Interface générée par WICALE 1.0	80
Figure 42: Architecture des composants logiciels de PILAF.....	83
Figure 43: Ajout du bouton d'édition simple de WICALE 1.1	87
Figure 44: Message de demande du mode d'ouverture.....	87

Figure 45: Exemple de l'éditeur simple Notepad et de l'éditeur par défaut.....	88
Figure 46: Navigation sous CASH.....	88
Figure 47: Trois étapes de la préparation des fichiers de navigation dans WICALE 2.0.....	89
Figure 48: Navigation sous WICALE 2.0.....	90
Figure 49: Architecture générale d'EMEU_w.....	91
Figure 50: Page HTML renvoyé par SECTra_w.....	93
Figure 51: Construction d'un compilateur avec l'outil de génération des compilateurs.....	95
Figure 52: Sortie de NOOJ.....	113
Figure 53: Annotation de texte par les systèmes-Q dans le projet OMNIA.....	113
Figure 54: Temps de traitement dans le projet OMNIA.....	113
Figure 55: Moniteur d'Ariane-Y en utilisant REXX/REXXstack.....	116
Figure 56: Exemple de sortie de VISULEX (anglais- français « BEX-FEX »), les commentaires et les codes sont extraits séparément et mis en regard.....	138
Figure 57: Interface d'UWGate.....	139
Figure 58: Interface de consultation en colonnes de PARAX.....	140
Figure 59: Exemple de dictionnaire de Yakushite.net.....	142
Figure 60: Exemple de présentation en « furcôide » du dictionnaire FEV.....	146
Figure 61: Organisation par acceptions interlingue (axies) dans NADIA.....	147
Figure 62: Interface d'édition d'une entrée générée par Papillon.....	149
Figure 63: Macrostructure de PIVAX.....	150
Figure 64: Exemple de volumes dans une base PIVAX.....	150
Figure 65: Données publiques et privées en PIVAX.....	152
Figure 66: entrée ' <i>ralentir</i> ' dans l'interface d'édition pour le volume UNL-FR.....	155
Figure 67: Interface de navigation d'un résultat de recherche.....	160
Figure 68: Interface de l'édition d'une entrée (1).....	161
Figure 69: Interface de l'édition d'une entrée (2).....	161
Figure 70: Interface de composition des critères de recherche avancée.....	162
Figure 71: Plusieurs profils possibles pour un utilisateur.....	162
Figure 72: Un profil avec les paramètres de présentation des colonnes.....	163
Figure 73: Segmentation multiple des occurrences.....	164
Figure 74: Interface d'envoi d'une requête d'IATE.....	165
Figure 75: Résultat renvoyé par IATE pour le mot (terme composé) « ground water ».....	166
Figure 76: Interface simplifiée de PIVAX dans SECTra_w.....	168
Figure 77: Exemple du mot ' <i>motor</i> ' dans une BDLMpT.....	172
Figure 78: Diagramme de cas d'usage de BDLMpT avec plusieurs volumes de données dans PIVAX.....	173
Figure 79: Diagramme de flot de travaux pour une opération simple dans IToldU.....	176
Figure 80: Recherche avancée par plusieurs critères.....	177
Figure 81: Le processus de manipulation directe des graphes lexicaux dans PIVAX.....	181
Figure 82: Exemple de graphe de MuLLinG.....	183
Figure 83: Interface de manipulation directe des graphes lexicaux par programmation en langage narratif (PIVAX_lgNarr).....	185
Figure 84: Présentation ergonomique de la sortie d'ATEF.....	211
Figure 85: Arbre entrée à SYGMOR.....	211
Figure 86: Arbre sortie de TRANSF.....	212
Figure 87: Exemple de transformation d'arbres décorés sous ROBRA.....	213
Figure 88: Editeur de LT.....	216
Figure 89: Exemple de LT pour la transcription de formes phonétiques.....	216

LISTE DES ABREVIATIONS

BDLM	Base de Données Lexicale Multilingue
BDLMpT	Base de Données Lexicale Multilingue préTerminologique
EDL	Environnement de Développement Linguiciel
LN	Langue Naturelle
LSPL	Langage Spécialisé de Programmation Linguicielle
K	x 1 000
M	x 1 000 000
MT	Mémoire de Traduction
IL	Interlingua
SI	Structure Interne
SA	Structure Abstraite
TA	Traduction Automatique
TAO	Traduction Automatisée par l'Ordinateur
TH	Traduction Humaine
THAM	Traduction Humaine Assitsée par la Machine
TALN	Traitement Automatique de Langues Naturelles

CONVENTIONS TYPOGRAPHIQUES

Style	Texte cité
Citation	Citation
Code source ou exemples	Code source ou exemples
<i>Algorithme</i>	Algorithme
<i>termes</i>	Terme utilisé dans un contexte ou un système

Introduction

L'intégration des systèmes de *Traduction Automatique (TA)* et des systèmes de *Traduction Humaine Assistée par la Machine (THAM)* au niveau de la préparation et de l'opération d'un système complet est un objectif formulé depuis les années 1990, mais il n'a encore été atteint par aucun système opérationnel, ni même par un prototype « préopérationnel ». La difficulté provient de grande distance entre leurs composants, tant linguiciels que logiciels.

Actuellement, on cherche à automatiser le développement des linguiciels (les connaissances lexicales, les corpus, les grammaires) dans toutes les techniques de TA. Cependant, cet effort aide à réduire certains coûts, mais on ne peut pas remplacer totalement les spécialistes (linguistes, lexicographes...) dans ce processus. Une autre approche consiste à proposer à des utilisateurs non spécialistes de contribuer. Mais cela est limité aux informations simples et demande des outils intuitifs.

Suite à la croissance de l'accès à l'information, on a besoin de nouveaux couples de langues, et de nouveaux systèmes de TA ou de THAM. La construction de tels systèmes n'est jamais partie de zéro, mais plutôt de composants existants. Il faudrait le réaliser en respectant l'extensibilité et en particulier l'ouverture possible aux manipulations humaines ?

Toutes ces demandes nous conduisent à la grande question : comment peut-on construire un système de *Traduction Automatisée par l'Ordinateur (TAO)* hétérogène dont les composants linguiciels et logiciels sont développés en divers lieux et avec les technologies différentes ? De plus, comment rendre un tel système ouvert à l'ajout de fonctions supplémentaires et à des interventions d'utilisateurs ayant des profils différents, tant sur la préparation des ressources linguistiques que sur la traduction elle-même (préédition, interaction, post-édition), avec apprentissage du système ?

Notre but est de résoudre cette question. Pour ce faire, il faut étudier quels sont les composants possibles et couvrir le mieux possible les divers aspects de la construction d'un système de TAO afin d'arriver à une méthodologie et à des principes adéquats.

Ces composants peuvent être divisés en (1) *composants logiciels* et (2) *composants linguiciels*. Selon l'architecture d'un système de TA(O), un composant logiciel peut être : (1.1) *un moniteur*, (1.2) *un moteur*, ou (1.3) *un Environnement de Développement Linguiciel (EDL)* du système. Dans cette thèse, on analyse et expérimente chacun de ces composants du point de vue de l'extensibilité et de l'intégrabilité.

Un composant linguiciel peut être: (2.1) des connaissances lexicales, (2.2) des connaissances « corporales » (corpus bilingues plus ou moins annotés), ou (2.3) des grammaires, automates et « programmes linguistiques » écrits dans des langages de programmations usuels ou quelles fonctions sont nécessaires ou simplement utiles pour leur développement.

Dans cette thèse, on vise à résoudre le problème de la construction d'un système de TAO hétérogène sur chacun de ces aspects, en vue d'arriver à la définition d'un système global.

Dans la première partie, intitulée « *Vers des systèmes de TAO hétérogènes* », nous exposons l'évolution vers les systèmes de TAO hétérogènes en distinguant 3 approches : l'intégration entre les systèmes de TA et les systèmes de THAM, la construction de systèmes hybrides à partir d'autres systèmes ou composants, et en particulier la synergie homme-machine.

Dans la deuxième partie, intitulée « *Méthodes pour la convergence TA ↔ THAM* », nous exposons des techniques de fusion entre la TA et la THAM. C'est d'abord l'intervention des non-spécialistes dans les composants de TA, puis l'intégration des techniques provenant de la THAM dans les systèmes de TA. Enfin, on présente l'application des composants linguiciels (les dictionnaires, les corpus, les grammaires) pour à la fois la TA et la THAM.

Introduction

La troisième partie, intitulée « *Gestion des bases de connaissances lexicales et intégration* », consacre le plus important des systèmes de TA à savoir leurs dictionnaires. Nous voyons comment construire une base de données lexicales multilingues PIVAX, capable de gérer les connaissances lexicales de plusieurs systèmes de TA, et pour l'utilisation commune entre TA et THAM.

Partie 1. Vers des systèmes de traduction assistée par l'ordinateur hétérogènes

Introduction

Nous présentons dans cette partie l'évolution des recherches et pratiques en Traduction Automatique (TA) vers des systèmes hétérogènes, et plus précisément leurs aspects logiciels. Nous cherchons à caractériser les niveaux possibles de l'hétérogénéité, les composants à utiliser en commun entre les systèmes, et l'architecture logicielle avec la technique de réalisation retenue.

Les différents composants des systèmes de Traduction Automatisée par l'Ordinateur (TAO) hétérogènes seront construits par des équipes différentes, distribuées autour de la planète, avec des méthodes (algorithmiques) et des outils (langages spécialisés ou LSPL, types de dictionnaires, de grammaires, de transducteurs à règles) différents, et des environnements de développement linguiciel (EDL) différents, un peu comme on construit les Airbus. La réutilisabilité des ressources et composants logiciels et linguiciels est le point essentiel dans le contexte de la construction de ce type de systèmes hybrides.

D'ailleurs, la traduction « démocratisée » [Boitet, 1999] a été confirmée grâce aux technologies avancées du Web. Il y a de plus en plus d'environnements basés sur le Web pour gérer ou développer des composants comme XTM [XML-INTL, 2009a] pour les mémoires de traductions, Google Translation Toolkit [Google-Translation, 2009a] pour la post-édition de TA, Yakushite.net [Yakushite.Net, 2008a] pour le développement lexical intégré à la TA, *etc.*

Pour construire un tel système, il nous faut avoir une vue claire sur (1) quelles sont les fonctions possibles à intégrer entre les systèmes, (2) quels sont les composants (logiciels et linguiciels) à partager, (3) quels sont les moyens nécessaires pour construire un système hétérogène à partir de fonctions et de composants existants.

Dans cette partie, on essaie d'étudier systématiquement ces aspects. Nous présentons d'abord un bref historique de la convergence des approches centrées sur la TA et sur la traduction assistée par la machine (THAM), tout à fait séparées de 1980 à 1990, vers la TAO hétérogène. Nous présentons ensuite quelques efforts récents de construction de systèmes de TA/TAO hétérogènes, afin d'arriver à caractériser la problématique de construction des systèmes de TA hétérogènes à partir des systèmes de TA/TAO homogènes. Enfin, nous détaillerons un exemple de système de TA complet, Ariane-G5, qui est en cours de réingénierie vers un système de TAO hétérogène complet dans le cadre du projet Ariane-Y.

Chapitre 1. TA et THAM

Introduction

Un système de TA est développé par des linguistes, et des informaticiens, avec une architecture opérationnelle et une architecture linguicielle généralement bien définies. Les modules informatiques (moteur, moniteur...) sont destinés à résoudre les problèmes de traitement des langues dans la traduction de la langue source à langue cible par la machine ou à opérer le système. Les ressources linguistiques (dictionnaire, grammaire, corpus) servent à définir des phénomènes linguistiques ou font partie du développement du système (test, validation...).

Un système de THAM est destiné à supporter le travail de traduction professionnelle. Les ressources (terminologie, mémoire de traduction) sont maintenues par les traducteurs, ou des experts dans un domaine.

Cette distinction freine l'intégration de ces systèmes. Tout ce qu'on sait faire en pratique est d'appeler la TA pour prétraduire le texte dans un système de THAM. Cependant, on a vu des efforts de fusion concrets dans un certain nombre de systèmes.

1.1 Ajout de fonctions de THAM à la TA

L'idée de la « TA du réviseur » est d'automatiser le « premier jet » de traduction, et de le faire réviser par un spécialiste humain. C'est pourquoi, dès que l'informatique est devenue interactive, on a intégré aux systèmes de TA des aides à la révision, sous diverse formes. Nous illustrons cela avec les systèmes Ariane-G5, PAHO, Pensée et BehaviorTrans.

1.1.1. Mode de révision en Ariane-G5

Le système Ariane-G5 [Boitet, 1993] créé par le GETA est un environnement complet et complexe de développement linguiciel pour la TAO. Un linguiste peut créer un système de TA sur Ariane-G5 sans le support d'un informaticien. Son environnement intégré de post-édition suit le même principe : l'utilisateur n'a besoin de connaître que les commandes d'Ariane-G5 et de l'éditeur XEDIT. Voici un exemple d'écran de révision du fichier BOULE du corpus LIDIA en entrant la commande : PRREVIS A0309 T0003 VT RU5 FR5.

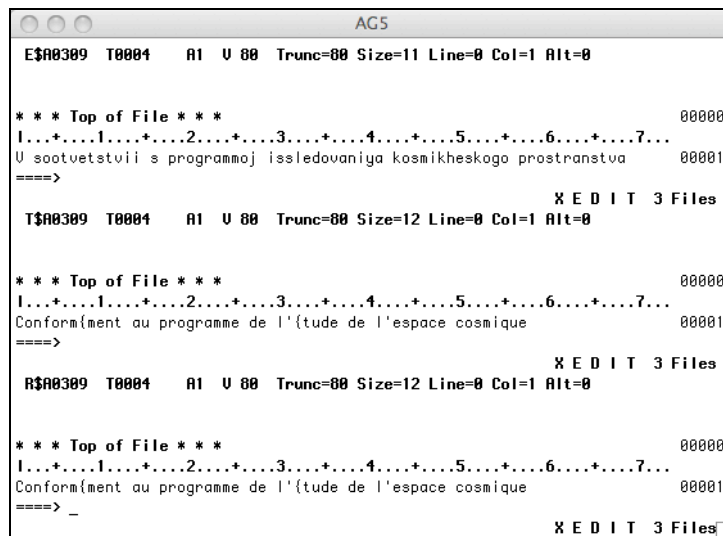


Figure 1: Un texte ouvert en mode « révision » sous Ariane-G5

Dans cet exemple, le système de TA RU5-FR5 (russe-français) fournit deux traductions de la phrase source russe « *V sootvetstvii s programmoj issledovaniya kosmicheskogo prostranstva* ». Ariane-G5 propose une interface d'édition en trois fichiers en parallèle dans XEDIT : le texte source, le texte traduit « *Conformément au programme de l'étude de l'espace cosmique* », puis le texte révisé « *Conformément au programme de l'étude de l'espace aérien* » initialisé par la TA. Pour présenter le résultat de post-édition, Ariane-G5 fournit une interface en colonne :

```
LISREVIS T RU5 FR5 A0309 T0003 N ETR C 1
```

-- TEXTE SOURCE --	-- TEXTE TRADUIT --	-- TEXTE REVISE --
17 noyabrya 1981 g v SSSR osuthestvlen zapusk okherednogo ISZ "Molnija-1", kotoryij prednaznakhen dlya obespekheniya yekspluatacii sistemy dalqnej	Le 17 novembre 1981 en URSS on a réalisé le lancement d'un nouveau satellite "éclair (AMBIGU: foudre -1)" destiné à l'assurance de l'exploitation du système de liaison	Le 17 novembre 1981 en URSS on a réalisé le lancement d'un nouveau satellite "éclair (AMBIGU: foudre -1)" destiné à l'assurance de l'exploitation du système de liaiso
		MORE... CLMP3000

Figure 2: Présentation en colonnes dans Ariane-G5

Ce système de TA a été utilisé dans plusieurs projets:

Table 1: Applications d'Ariane-78 puis Ariane-G5

Systèmes opérationnels	
russe-français GETA (DRET), 1978-1987	bulletins signalétiques du VINITI
français-anglais B'VITAL/SITE, 1983-1992	manuels de maintenance aéronautique
Prototypes	
allemand-français GETA (DRET)	1979-1986
anglais-malais GETA-USM	1980-1985
français-anglais GETA (DGT)	1981-1982
déconvertisseur (UNL-FR)	1996-
enconvertisseur (FR-UNL)	1999-
Maquette pédagogique	
anglais↔français GETA (ADI)	1982-1983
Maquettes de recherche	
portugais-anglais; anglais-thaï; anglais-japonais; anglais-chinois; chinois-5 langues; français-chinois	1978-1987
projet LIDIA (français → russe, allemand et français, avec désambiguïsation et rétrotraduction)	1989-1995
maquette de déconvertisseur (UNL-Chinois)	2000-2004

1.1.2.PAHO

Le déploiement de la TA à la PAHO [Aymerich, 2004; Leon, 2000] (Pan American Health Organization) a commencé en 1980 avec un volume de 4M mots par an sur le couple de langues anglais↔espagnol (1980-1985). Le premier système a été SPANAM-1 pour l'espagnol-français, écrit en PL/1, puis en 1982 on a construit ENGSPAN pour la traduction de l'anglais vers l'espagnol. Ces deux systèmes noyaux ont été réécrits en C en 1992.

Récemment, on a construit deux nouveaux couples anglais↔portugais (2003) et espagnol↔portugais (2004). La dernière version en 2008 est la v4.3 avec support de dictionnaire, traduction collaborative via réseaux LAN, et intégration avec MS Office.

Les traducteurs reçoivent des requêtes de traduction des documents dans le contexte défini médical (formulaire, manuel, rapport...). Le TTS (Translation Tracking System) permet de suivre le flot de travaux de cette requête (Figure 3). Quand un segment à traduire arrive, le système intégrant TA et THAM cherche d'abord dans la mémoire de traductions (WordFast); s'il trouve un segment (dans seulement 5% des cas), il le rend tout de suite aux traducteurs, sinon il appelle PAHO MTS (Machine Translation System) pour le traduire.

L'usage de la TA à la PAHO (et également à l'OMS, l'Organisation Mondiale de la Santé à Genève depuis 1995) s'applique à 90% des cas de traduction (250K mots/mois, soit 1000 pages standard) et réduit de 30% le coût de traduction. Le traducteur peut donner des retours (feedbacks) pour les termes non traduits ou nouveaux, ces termes sont importés dans les dictionnaires des systèmes de TA par un outil de fusion de dictionnaires.

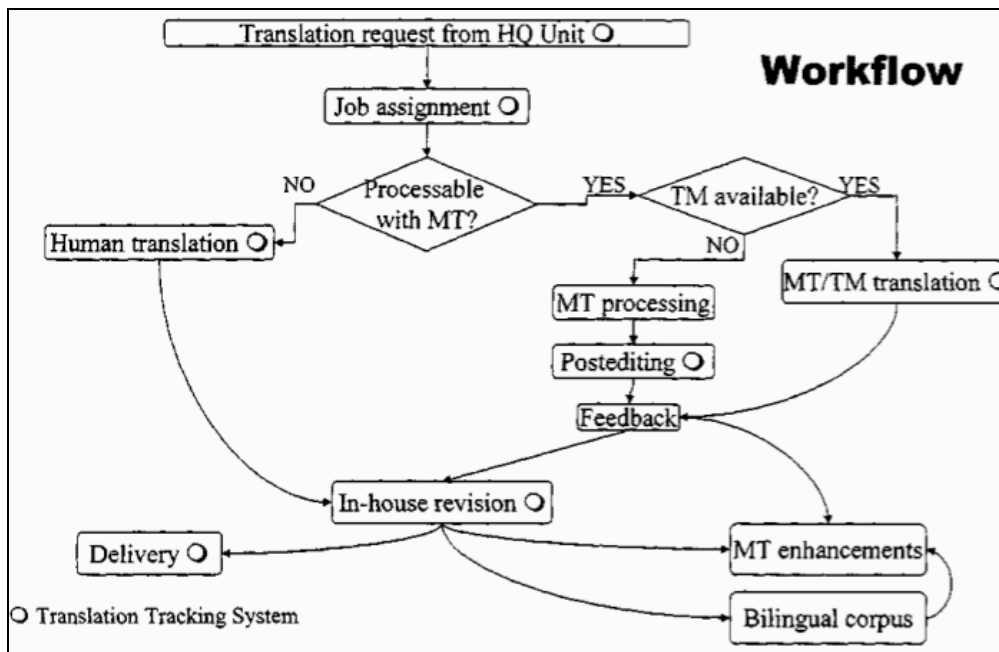


Figure 3: Flot de travaux dans le processus de traduction à PAHO

1.1.3.Pensée, Yakushite.net

Le système de TA Pensée [Shimohata et al., 1999] a été commercialisé en japonais-anglais en 1986 et en anglais-japonais en 1988. C'est l'un des premiers systèmes de TAO intégrant directement une contribution lexicale humaine aux ressources lexicales utilisées par un système de TA. La communauté peut contribuer avec des termes ou des schémas liés à des termes via un environnement Web (Yakushite.net) [Murata et al., 2003; Yakushite.Net, 2008b]. Après révision, ils sont intégrés directement dans le système de TA Pensée. Ces données (1K de schémas et 130K de formes fléchies) sont stockées dans une base lexicale.

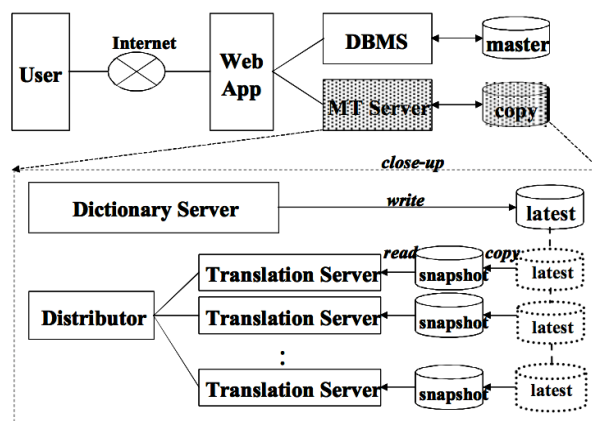


Figure 4: Liaison entre la base lexicale yakushite.net et le système de TA Pensée

1.1.4. BehaviorTran de Keh-Yih Su (Taipeh), amélioration de la TA par le retour des traducteurs

Dans le système de TA BehaviorTran (anglais-chinois) développé à l'Université de Tsing-Hua (Taipeih) en 1985, puis commercialisé en 1988, Keh-Yih Su [Su, Wu et Chang, 1992] a proposé un modèle qui permet de prendre compte des retours pour ajuster certains paramètres d'un système de TA. Le principe de ce mécanisme est qu'à partir des connaissances linguistiques statiques, on observe les sorties et on modifie certains paramètres du système comme le choix meilleur en langue cible dans le transfert par un modèle de probabilité pour choisir « les meilleurs styles » dans la traduction. Cela permet de créer plus facilement des systèmes de TA spécifiques.

L'avantage de cette méthode est de ne pas demander de travail supplémentaire au traducteur (pas de rapport, ni d'écran supplémentaire sur l'interface de traduction).

1.2. Ajout de composants de TA à la THAM

Le but de l'intégration de composants de TA dans des outils de THAM est de faciliter le travail du traducteur.

1.2.1. Intégration d'aides dictionnaires pour la THAM

1.2.1.1. Aide pure (dico « main gauche » d'Alain K. Melby)

Dès 1980, Alan K. Melby (BYU, Brigham Young University) a montré de façon opérationnelle qu'une des plus importantes aides dans la THAM était la consultation de dictionnaires. Il a proposé ce support « pur » et indispensable comme premier niveau de support. Il s'agit d'un accès au dictionnaire électronique actif à côté (dictionnaire « main gauche ») de l'interface de traduction. La base dictionnaire est préremplie à partir des lemmes existants dans le texte à traduire. On peut considérer cette aide comme un premier niveau de traduction automatique « mot à mot ».

1.2.1.2. Système de THAM Transactive (ALPS)

Le système ALPS (Automated Language Processing Systems) à Provo, Utah, USA a été créé en 1980 par des membres du groupe linguistique de BYU. C'est un des premiers systèmes de TAO interactifs sur PC (IBM AT). Il contient aussi un logiciel de traitement de textes multilingue avec deux colonnes pour les langues source et cible. Tous les styles sont copiés automatiquement.

ALPS fournit d'abord une aide de consultation des mots (AutoTerm) d'un dictionnaire bilingue d'un texte en cours de traduction. Si on ne trouve pas un mot, on peut l'ajouter. Le panneau de référence au dictionnaire est toujours en bas de l'interface.

Une autre fonction utile est la réutilisation de traductions déjà faites. Il y a un fichier qui contient les textes déjà traduits. Sur une unité de traduction (qui peut être une phrase ou une unité syntaxique), le traducteur peut extraire des morceaux de texte déjà traduits précédemment et insérer leurs traductions. Cela signifie que le système est muni d'une mémoire de traduction locale.

Transactive offrait de nombreuses autres fonctionnalités comme l'intégration d'un éditeur multilingue, la gestion des documents, gestion des traductions, la consultation des termes, un logiciel de traitement de texte intégré et un éditeur de révision (ALPS Translation Editor). ALPS était un des meilleurs du marché des systèmes de THAM/TA à cette époque (1980-1985), puis la compagnie a changé d'objectif et est devenue un fournisseur de services de traduction (ALPNET), utilisant ses outils internes, sans les commercialiser.

1.2.1.3. Système de THAM de Weidner

À partir de 1977, Weidner Communications Corporation (WCC) [Perscheid, 1985] a investi pour construire des systèmes de THAM appelés microCAT (mono-utilisateur) et macroCAT pour plusieurs utilisateurs travaillant à distance, qui ont dominé le marché à cette époque.

Une fois que les textes à traduire sont entrés dans système, le système réalise leur analyse morphologique et compare les occurrences dans le dictionnaire. Dès que les mots inconnus sont identifiés, ils sont envoyés aux lexicographes pour qu'ils complètent les dictionnaires au fur à mesure du processus de traduction.

Les textes sont prétraduits par la machine (6K à 8K mots/heure soit 24 à 32 pages). Les calculs de recherche de mots et de traduction sont faits à l'avance. La tâche de post-édition consiste en: la correction grammaticale, la vérification sémantique, la réinsertion des balises de formatage. Au cours de la traduction, le traducteur peut toujours consulter un dictionnaire en ligne pour trouver des équivalents. Le temps moyen de post-édition est 600 à 1 200 mots par heure d'un traducteur (2,4 à 4,8 pages standard). Au fur et à mesure, le traducteur peut donner des feedbacks au système de TA via sa contribution au dictionnaire.

1.2.2. Ajout de mémoires de traductions

Le deuxième niveau de support au traducteur est offert par les mémoires de traduction de « segments », unité de traduction à sens complet (phases ou titres).

En 1978, Peter Arthern a proposé l'idée d'utiliser une « archive de traductions » qui stocke les phrases source et les phrases traduites afin de pouvoir retrouver facilement une traduction. En 1980, Martin Kay a décrit un système de MT¹ avec des tâches réalisables sur la machine comme l'édition de texte et la consultation de dictionnaire, qui amélioreraient la productivité des traducteurs. En 1982, A. Melby a relié les idées des dernières années, et a proposé des aides aux traducteurs à trois niveaux [Melby, 1982]. Cette synthèse incorpore l'idée de l'alignement, qui est un élément très important dans la conception des systèmes de MT. Il y a des systèmes de MT qui sont nés à cette époque : ALPS (1980), ETOC (1988).

À partir de 1990, on assiste au développement de nombreux systèmes de MT commerciaux, dont:

- TRADOS, né par une compagnie allemande de traduction professionnelle ;
- DéjàVu, outil de MT développé par Atril ;
- Transit, développé par STAR (Suisse, Allemagne, Russie, France);

¹ Mémoire de traduction

- TM/2, développé par IBM.

Angelica Zerfaß [Zerfaß, 2002] et Celia Rico [Rico, 2000] ont proposé des comparaisons des systèmes de MT et des guides pour choisir un système de MT adapté à un environnement.

Actuellement, il y a environ 20 compagnies actives qui commercialisent des outils de MT et il y aurait 2,725 milliards d'unités de traduction (segments) vérifiées et spécifiées au total dans l'industrie (d'après *Translation Memory Marketplace* [Marketplace, 2008]).

L'évolution des systèmes et ressources de MT est dirigée vers une architecture centralisée et fondée sur des formats d'échange de XML: TMX (Translation Memory Exchange Format), TBX (Termbase Exchange Format), SRX (Segmentation Rules Exchange Format), OLIF (Open Lexicon Interchange Format), XLIFF (XML Localization Interchange File Format).

Ces outils proposent essentiellement des fonctions de gestion de traduction et de post-édition, ainsi que la consultation terminologique. La spécification de format standard de stockage de ressources de MT (ex. LISA) entre les systèmes assure l'homogénéité et l'échange facile entre les systèmes.

1.2.2.1. Systèmes usuels de gestion de MT

1.2.2.1.1. TM/2 (IBM)

Pour faciliter la réutilisation des traductions existantes, IBM a construit dès 1970 au GDSL (IBM's German Software Development Lab) le système Translation Manager abrégé en TM/2 [IBM-Corp, 1993] car il tournait sous OS/2. Depuis quelques années, il tourne sous plusieurs plates-formes, et a été renommé TM (Translation Manager). Il est utilisé par tous les traducteurs travaillant pour IBM, mais il n'est plus commercialisé.

Après avoir été importé dans TM/2, le fichier doit être segmenté en unités de traduction (phrases ou titres), puis TM/2 vérifie s'il existe des segments présents dans la MT. La traduction est faite sous un éditeur propriétaire intégré à TM/2 (Figure 5).

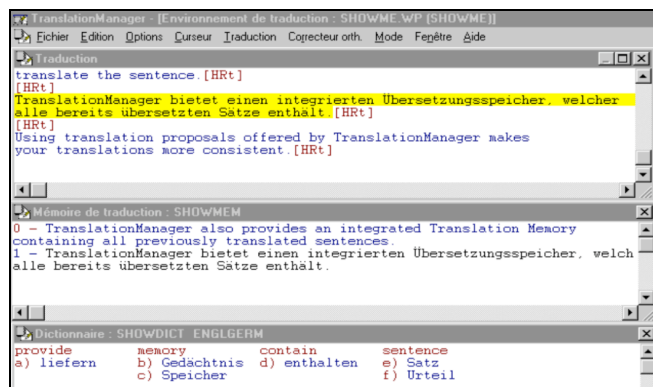


Figure 5: Interface de traduction de TM/2

Dans l'organisation interne, la mémoire de traductions est constituée d'un fichier SGML crypté. Ce fichier contient une suite de segments bilingues avec un identificateur en tête de chaque segment.

Chez IBM, il y a aussi un système de TA, LMT (disponible depuis 1984). Il y a eu des projets pour utiliser les deux systèmes de façon intégrée, mais cela s'est révélé impossible parce qu'il y a une séparation entre leurs dictionnaires et que les développeurs de la TA sont très loin des utilisateurs (traducteurs) et n'accordent aucune priorité à leurs demandes. La TA devient de moins en moins bonne au fur et à mesure que la terminologie évolue, tandis que la mémoire de traductions et le lexique personnalisable intégrés à LMT deviennent de plus en plus utiles.

1.2.2.1.2. Trados (SDL)

La société Trados [TRADOS, 2008] commercialise un outil de THAM appelé SDL Trados Workbench, qui a environ 200 000 utilisateurs. Cet outil se compose d'un gestionnaire de terminologie (Multiterm), un module d'alignement TAlign (sous DOS) ou WinAlign (sous Windows) qui permettent de former la MT utilisée dans Workbench. Il y a un produit pour les traducteurs individuels (SDL Trados Suite Freelance) et pour les traducteurs travaillant en groupe (SDL Trados Suite Professional).

Pour utiliser Trados dans un projet de traduction, il faut définir la MT et la terminologie pour ce projet.

Au cours de la traduction, un traducteur peut travailler en deux modes : prétraduction et traduction interactive. Le premier mode groupe tous les documents sous Microsoft Office pour que le traducteur puisse les réviser et les compléter. Le second mode permet au traducteur de profiter de toutes les informations existant dans la MT et dans la base terminologique. Une fois un document importé dans Trados, Trados propose une autre phrase assez proche trouvée dans la MT selon le seuil défini par l'utilisateur. Il peut fournir un rapport des segments non trouvés. Chaque fois qu'un utilisateur post-édite un ou des segments, il peut demander de retraduire tout le document pour appliquer les nouvelles traductions.

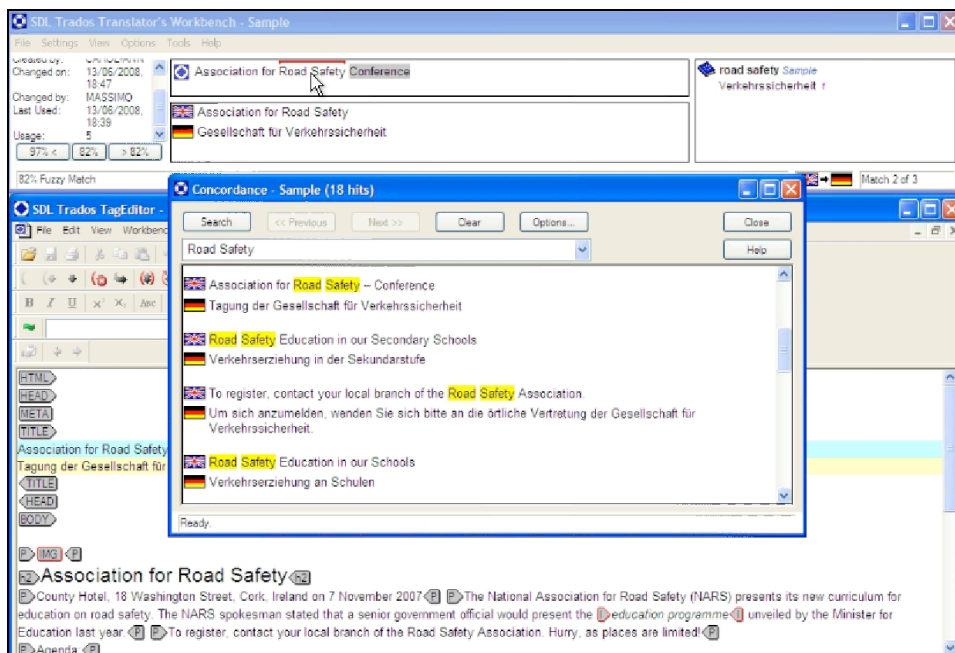


Figure 6: Capture d'écran dans le mode de traduction interactive de SDL TRADOS

La traduction automatique n'est pas intégrée à Trados. Pour un nouveau segment, Trados cherche (en mode exact et flou) dans la MT pour remplir la traduction. Cependant, Trados supporte des formats standard comme TMX, TBX... Sur ces formats, on peut appeler des systèmes de TA usuels (ex. SYSTRAN) pour prétraduire les segments et les importer dans Trados.

1.2.2.1.3. Déjà Vu (ATRIL)

La société ATRIL [DéjàVu, 2008] a créé un outil de THAM nommé DéjàVu. Comme la plupart des autres outils, il propose un gestionnaire de projets, un flot de travaux, des listes terminologiques et une MT. Pour la prétraduction, DéjàVu utilise la recherche approximative des phrases « proches » dans la mémoire de traduction existante.

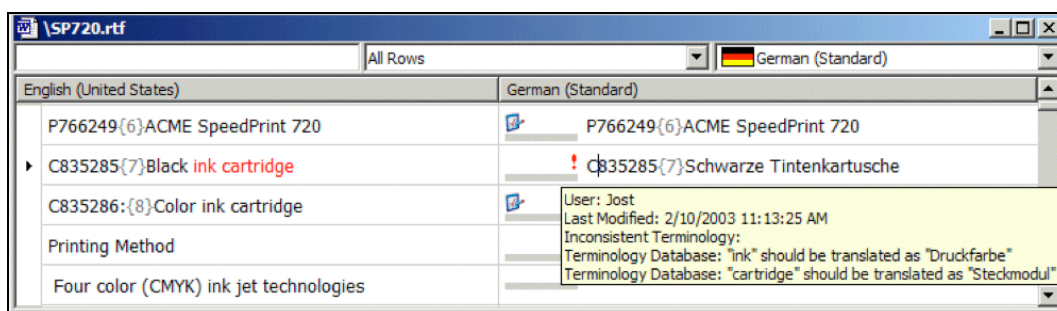


Figure 7: Capture d'écran de l'environnement de traduction de DéjàVu

Le flot de travaux dans la traduction est pratique. Si sur chaque étape, tâche de flux, DéjàVu exporte un format connu avec toutes les informations de MT, trace de traduction..., on a possibilité d'insérer des traitements comme l'appel de notre système de TA, prise en compte de dictionnaire d'utilisateur ou l'utilisation d'une MT spécialisée sur ce flux.

Comme Trados, DéjàVu n'a pas la TA pour prétraduire les documents.

1.2.2.1.4. Transit (STAR)

Cet outil a une interface unique pour travailler avec plusieurs types de fichiers différents avec un éditeur commun propriétaire.

Dans la phase de segmentation, on peut définir des règles (expressions régulières) pour segmenter le document en unités de traduction. Une fois le fichier segmenté, la phase de prétraduction insère les segments trouvés dans la mémoire de traduction grâce au module « Réseau Associatif ». Transit groupe tous les segments non traduits dans un fichier séparé. L'interface d'environnement de traduction est (Figure 8²)

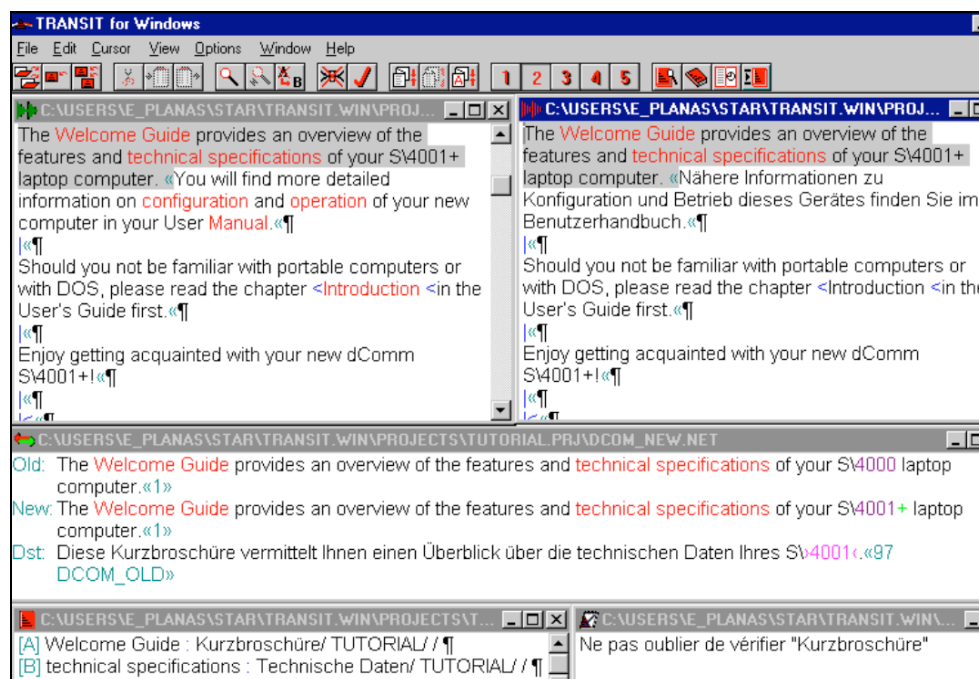


Figure 8: Interface de traduction de Transit

Le point fort de Transit est pouvoir traiter plusieurs types de fichier. Dans le contexte d'intégration d'un tel système, on peut insérer des codes non traitables par Transit dans le

² La capture d'interfaces est extraite de la thèse d'Emmanuel Planas.

fichier d'entrée, récupérer la sortie de Transit puis comparer des segments non traduit, et appeler la TA sur ces segments.

1.2.2.2.XTM (XML-INTL), système de MT sur le Web

Le société XTM-INTL développe un système de MT fonctionnant totalement en ligne [XML-INTL, 2009b]. Il supporte la contribution lexicale au cours de la post-édition, la fonction de chercher/remplacer dans tout un document ou dans toute la mémoire de traductions, la gestion de flot de travaux, et l'import/export en format XML des résultats de traduction (voir Figure 9).

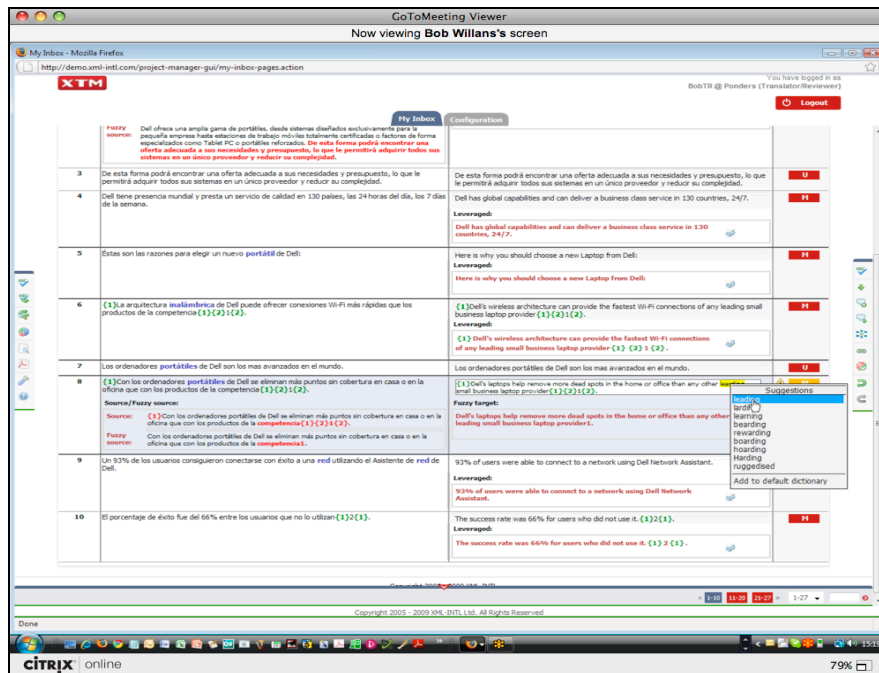


Figure 9: Écran d'interface de post-édition avec recherche de dictionnaire intégrée

L'avantage fort de XTM est que le système fonctionne totalement sur Web. On a une grande facilité pour intervenir le système par échange de commandes, données sous forme d'une page HTML et HTTP.

1.2.2.3.Similis (Lingua&Machina), nouvelle génération de MT

Créé par É. Planas en 2004, Similis est un logiciel professionnel d'aide à la traduction destiné aux traducteurs indépendants, aux agences de traduction et aux traducteurs en entreprise. Il est le premier de la deuxième génération de systèmes de MT [Planas, 2005] avec l'analyse morpho-syntaxique de parties de phrase (chunks), le stockage dans la mémoire de traduction d'unités de traduction sous-segmentales. Par exemple, « *Les mémoires de seconde génération changent le monde de la traduction* » traduit en « *Second generation memories change the translation world* », la correspondance entre les groupes nominaux « *mémoires de seconde génération* » et « *second generation memories* » est possible.

Pour un nouveau projet, Similis analyse des traductions antérieures pour déduire une mémoire de traductions adaptée. Lors de l'alignement, on peut extraire des termes dans un document, et il y a un dictionnaire « main gauche » des termes de base donné par Similis.

Segments sources (235)	Position	Score	Position	Segments cibles (230)
Another factor which makes improved macro-economic co-ordination a matter of urgency is the globalisation of financial markets, which is proceeding with accelerating speed, but which reflects less and less the turnover of goods and services.	95(7)	95	91(8)	Un autre élément qui rend urgente l'amélioration de la coordination réside dans la mondialisation des marchés financiers laquelle s'opère à une vitesse croissante mais reflète de moins en moins le mouvement des biens et des services.
This suggests that interest rate modifications have a main impact on the price of financial	95(8)	100	91(9)	Cela suggère que les variations des taux d'intérêt ont une influence notable sur le prix des biens
Chunks sources	Position	Score	Position	Chunks cibles
the globalisation of financial markets	5	50	7	la mondialisation des marchés financiers
accelerating speed	9	50	12	une vitesse croissante
the turnover of goods and services	13	75	15	le mouvement des biens et des services

Figure 10: Exemple de l'extraction de parties de phrase (chunks) par Similis à partir d'un document source et de sa traduction

Dans la structure de MT, Similis utilise le modèle TELA (voir 1.2.3.2.3.1). Pour un document d'entrée, après l'indexage, on fait l'alignement des segments par le calcul de distance entre des segments sur le modèle TELA. On ne garde pas le résultat de calcul dans le mémoire, donc on peut adapter la MT pour un domaine spécialisé.

La structure de Similis permet de recycler plus de traductions dans la MT et nous donne la possibilité de créer une TA spécialisée.

Similis fonctionne sous Windows avec un éditeur intégré XEditeur pour traiter plusieurs types de fichiers : HTML, MS Word, XML, SGML...

1.2.3. Ajout ou intégration de TA à des systèmes de THAM

1.2.3.1. Appel de systèmes de TA « experts »

1.2.3.1.1. Intégration de LMT à TM/2 (IBM)

Avec TM/2, IBM possède également un système de TA, LMT (Logic-based Machine Translation System) depuis 1984 [McCord, 1989]. Il est prévu depuis très longtemps de pouvoir appeler LMT dans une phase de « prétraduction » de TM/2 pour les segments non trouvés dans la mémoire de traduction, mais cela n'a jamais été réellement utilisé, à cause du manque de synchronisation entre les dictionnaires de TA et ceux de TH.

Il faudrait en effet que le système LMT connaisse la terminologie utilisée dans chaque projet de TM/2 pour traduire convenablement. Mais les développeurs du système LMT étaient (et sont) séparés utilisateurs de TM/2 par de grande distance, et surtout par des barrières administratives : les dictionnaires techniques de TM/2 deviennent vite obsolètes par rapport à ceux des traducteurs.

1.2.3.1.2. Projet EuroLang Optimizer avec appel de LOGOS, METAL et Ariane-G5

Le projet EuroLang Optimizer (EuroLang³) fournissait une aide de traduction à partir de six langues sources (français, anglais, allemand, espagnol, italien, néerlandais) vers onze langues cibles (les 6 langues sources plus danois, finnois, norvégien, portugais et suédois). À l'usage, on a mesuré un gain de temps de 30% à 70% par rapport à celui de la traduction traditionnelle en appelant la TA. Au début, on avait prévu d'intégrer des systèmes de TA (LOGOS, METAL et Ariane).

³ <http://www.eurolang.fr>

En 1995, on a construit une version adaptée fonctionnelle pour l'intégration avec le système LOGOS (Logos-Optimizer). Pour des raisons commerciales, on a pris LOGOS comme système de TA. Pour un segment donné, on cherche d'abord dans la MT. S'il n'y est pas, on appelle LOGOS pour traduire et on propose la suggestion sur les termes techniques contenus dans ce segment. Cela maximise l'utilisation de la MT dans le système.

Cette alliance a permis à EuroLang de renforcer la puissance de EuroLang Optimizer et de se concentrer sur les aspects de support de traduction. Et cela aide LOGOS à approcher le marché des systèmes de TA tournant sur PC. La synchronisation entre la base terminologique d'EuroLang et les lexies dans Logos était prévue. Mais EuroLang a été arrêté quand la société Site/EuroLang a perdu un appel d'offre important de la CEE face à TRADOS.

1.2.3.2.Perspectives: création d'un système de TA à partir des ressources de THAM

Avec la croissance importante des ressources de THAM (milliards de segments dans les MT), il est tendant créer un système de TA à partir de ces ressources. Pour le faire, on utilise bien autant l'approche probabiliste que l'approche experte. On va voir dans cette partie les méthodes de construction de tels systèmes.

1.2.3.2.1.Méthodes empiriques pour la création de systèmes de TA

1.2.3.2.1.1.TA probabiliste

L'idée de TA a été proposée en 1949 par [Weaver, 1949]. L'idée était alors de considérer la TA (russe-anglais) comme un « décodage » d'un message originellement anglais, qui aurait été « codé » en russe.

En 1990, cette approche a été réintroduite par le premier modèle de système de TA statistique [Brown et al., 1990] au centre de Recherche Thomas J.Watson d'IBM. L'idée était de traiter la traduction comme un problème de décryptage par la probabilité d'existence des chaînes de langue source (f) vers chaînes de langue cible (e) : $p(e|f)$. Le théorème de Bayes établit que : $p(e|f) = \max_e p(f|e)p(e)$, donc la traduction peut être formulée :

$$\tilde{e} = \arg \max_{e \in e^*} p(e|f) = \arg \max_{e \in e^*} p(f|e)p(e)$$

$p(f|e)$ est la probabilité que la chaîne (f) se traduise en chaîne (e) pour un corpus bilingue donné et $p(e)$ est la probabilité d'occurrence de la chaîne (e) dans la langue cible. Il y a des modèles variés basés sur les mots [Brown et al., 1993], les constituants [Wu, 1997], [Alshawi, Buchsbaum et Xia, 1997], [Kengo, Sato et Nakanishi, 1998] ou sur les contextes [Hermjakob et Mooney, 1997].

Au niveau des outils et systèmes réels pour la TA probabiliste, on peut distinguer les outils de préparation, les boîtes à outils complètes, les systèmes réels, et les scripts d'évaluation :

- Outils de préparation
 - Outils de création de corpus : TextSTAT, AntConc, WordSmith (pour les corpus monolingues ou comparables), ParaConc (pour les corpus parallèles)
 - Giza++ [Och et Ney, 2003] : outil d'entraînement des modèles d'IBM
 - Uplug [Tiedemann, 1999] : outil d'alignement aux niveaux : mot, morceaux (chunk) et phrase
- Boîtes à outils complètes
 - Pharaoh [Koehn, 2004] : décodeur pour le système de TA statistique basé sur phrases

- Moses [Moses, 2009] : outil de création d'un système de TA statistique avec un décodeur, un outil d'entraînement et de réglage pour tous les couples de langues ayant le corpus bilingue.
- Language Weaver [Language-Weaver, 2009] : boîte à outils commercialisée basée sur l'approche statistique.
- Systèmes réels
 - CANDIDE [DellaPietra et DellaPietra, 1994]
 - Google Translation [Google-Translation, 2009b]
- Scripts d'évaluation : BLEU, NIST, ORANGE, METEOR, ROUGE, WER, mWER

1.2.3.2.1.2.TA par l'exemple

L'idée de TA par l'exemple a été proposée par M.Nagao en 1984 [Nagao, 1984]. Selon lui, le principe de TA par l'exemple est la traduction par analogie basée sur un corpus bilingue de textes parallèles [Satoshi Sato et Nagao, 1990]. En fait, il s'agissait plutôt de TA par similarité (faisant intervenir 2 énoncés source, celui à traduire et un exemple similaire déjà traduit). La « vraie » traduction par analogie a été introduite par Y.Lepage [Y. Lepage et Denouel, 2005] et fait intervenir 4 énoncés source (celui à traduire et 3 autres) entre les quels existe une relation – toujours quaternaire – d'analogie. Quelques systèmes opérationnels de TA par l'exemple :

- Gaijin [Veale et Way, 1997]
- EDGAR [Carl, Way et Schäler, 2002]
- ReVerb [Collins et Cunningham, 1996]
- Guvenir & Cicekli [Cicekli et Guvenir, 2001]
- HPA/HPAT [Imamura, 2001] : TA par l'alignement des phrases en hiérarchie. Évaluation : 70% de traductions acceptées dans domaine de tourisme avec 170K phrases d'entraînement
- EBMT à Penang [Al-Adhaileh et Kong, 1999] : système de TA par l'exemple avec annotation sur la correspondance d'arbre-chaîne.

Pour construire directement un système de TA à partir des ressources de THAM, le processus est :

- Création d'un préprocesseur (souvent par l'approche experte) pour la langue source (codage, segmentation, lemmatisation, normalisation),
- Compilation des données (tables des différentes « modèles » à partir du corpus. Le temps de calcul non parallélisable est souvent longue sur une grande quantité de données (Moses prend des jours sur un PC pour créer un décodeur à partir du corpus WMT08 de 55K phrases).

On peut créer également certains composants d'un système de TA grâce au modèle de langue provenant de l'approche probabiliste comme les dictionnaires bilingues dans des systèmes de TA statistiques « experts ».

1.2.3.2.2. Avancées dans la TA « experte » avec dictionnaires et corpus communs

Les corpus utilisés en TA de l'écrit et de l'oral ont évolué, depuis les suites de test et les corpus d'essai des débuts, vers des corpus parallèles bilingues ou multilingues, bruts ou enrichis par des métadonnées et une grande variété d'annotations linguistiques. Ils sont assez petits et peuvent avoir une « granularité » importante en TA « experte », classique, mais sont énormes et de

granularité faible en TA « empirique », statistique ou fondée sur les exemples [Boitet, 2007].

Un défi actuel est d'unifier et de mutualiser la gestion et la construction des corpus de TA afin de pouvoir avoir un ensemble de corpus spécifiques à des domaines utilisables en TA « empirique ». Avec une base lexicale centrale et mutualisée, on pourrait créer de tels systèmes, avec un coût relativement faible, mais utilisables dans des domaines spécifiques.

1.2.3.2.3. Création d'un système de TA « expert » à partir des ressources d'un système à MT

Parmi les idées actuelles sur l'implémentation d'un système de TA par l'exemple, nous en retenons deux. L'une est de partir des ressources de type « chaîne » comme Lingua&Machina le fait pour le système de MT Similis [Machina, 2008], l'autre est de partir de ressources arborescentes, comme proposé par Ch.Chenon [Chenon, 2005].

1.2.3.2.3.1. Similis, transformation de THAM en TA par l'exemple

Dans le but d'améliorer l'utilisation des MT, Emmanuel Planas propose de les analyser et de les structurer en « étages ». C'est le modèle TELA [Planas, 1998]. La Figure 11 montre un exemple de MT TELA à 7 étages.

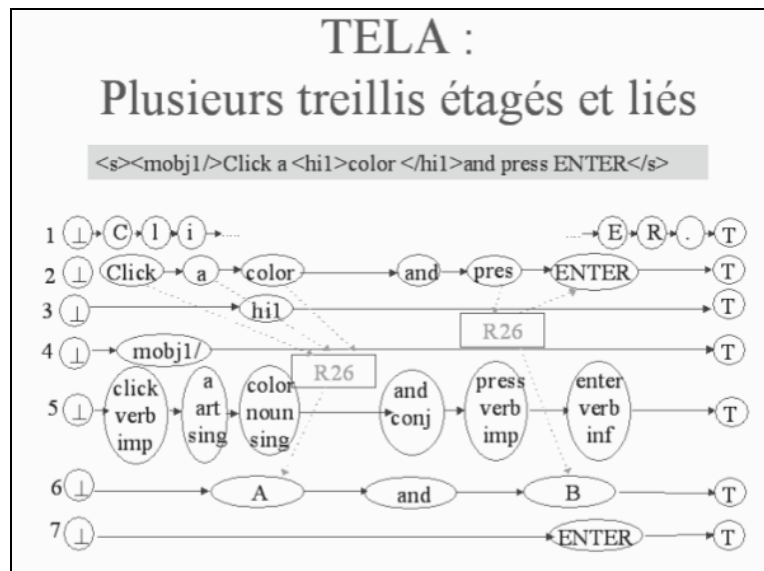


Figure 11: 7 étages de TELA

Chaque étage⁴ est un treillis, dont les nœuds correspondent aux objets à représenter pour cet étage. La notion de séquentialité est donc conservée et permet de voir les nœuds d'un chemin comme autant de successeurs. Les étages sont caractérisés par la profondeur d'analyse qu'ils représentent (caractères et entités, balises, texte comme suite de formes, lemmes, termes et fragments fréquents).

La correspondance entre les nœuds de chemins différents d'un même treillis ou entre deux treillis est donnée par des échelles linéaires. Quand il s'agit de relations complexes, comme par exemple la correspondance entre deux mots discontinus et un lemme, les liaisons permettent de les spécifier. L'ensemble des liaisons définit alors une relation de « correspondance » sur l'ensemble des treillis.

⁴ Nous citons l'explication donnée par E.Planas dans sa thèse [Planas, 1998].

Conclusion

Après ce chapitre, on a étudié le phénomène de synergie entre TA et TH(AM). L'intégration des techniques de THAM dans des systèmes de TA rend le service de TA plus applicatif dans le contexte d'utilisation. L'intégration des composants provenant de la TA à la THAM rend le travail des traducteurs professionnels plus efficace. En plus, avec des ressources relativement suffisantes de THAM, on pourrait créer un système de TA experte avec un coût moins cher et modifiable.

Cette unification nous dirige vers une classe de systèmes de TA et TAO hybrides constitué de composants intégrés par plusieurs auteurs sur plusieurs niveaux. La technique et l'architecture logicielles de réalisation de tels systèmes nous intéressent, plus particulièrement dans le contexte où l'accès à l'information est ouvert et facile. Cette étude est en synergie avec la tendance de génie logiciel actuel.

Chapitre 2. Évolution vers les systèmes de TA hétérogènes

Introduction

La construction d'un système de TA de haute qualité est très coûteuse. Les systèmes de TA se limitent alors à atteindre un niveau « acceptable » en sortie, et la qualité de la TA est souvent mauvaise. Cependant, dans un domaine restreint comme la traduction des « brèves » de la bourse de Tokyo (ALTFLASH), ou des bulletins météo (MÉTÉO), la TA donne un très bon résultat et ce résultat demande peu ou pas de correction par des humains.

Pour généraliser la TA à plusieurs domaines ou l'adapter à plusieurs projets de traduction. On cherche à combiner la sortie de plusieurs techniques de TA (TA basée sur les connaissances, TA empirique...), et chaque technique donne un résultat, qui est peut-être bon pour un domaine ou dans un contexte. À la sortie du système, on cherche à les combiner le mieux possible. Cela donne des systèmes « multiples ».

Une autre réalité est qu'il est très coûteux de construire un gros système de TA fortement multilingue. Dans le contexte actuel, on cherche à partager la construction d'un système entre plusieurs partenaires dans divers projets (MMT du CICC, UNL). Cela nous donne une seconde classe de systèmes hétérogènes.

Une autre approche est de séparer la partie informatique et la partie de connaissances, et de permettre aux utilisateurs d'intervenir sur certains niveaux lors du traitement sur la partie de connaissance selon le domaine à traduire. L'intervention des connaissances est faite sous un environnement adéquat. L'hétérogénéité ici est que le système ouvre aux utilisateurs (avec des profils différents) de participer à développer de ses composants.

Nous présentons donc maintenant quelques exemples de systèmes multiples, de systèmes à processus hétérogènes, et de systèmes à synergie homme-machine.

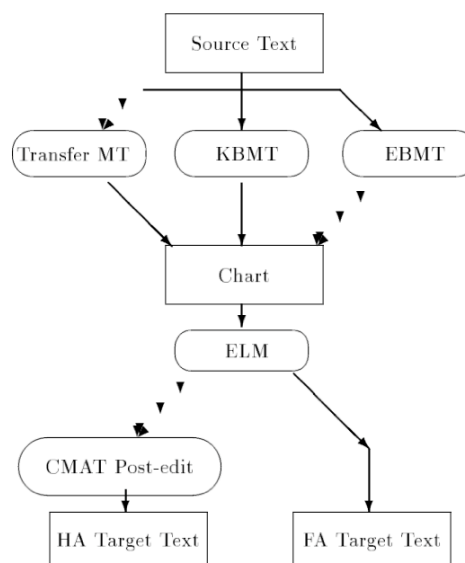
2.1. Systèmes de TA « multiples »

2.1.1. Exemples de systèmes connus

2.1.1.1. Pangloss (1995-1996), un système multiple avec même support et format informatique

Pangloss [Frederking et al., 1994] est un prototype de TA multi-moteur espagnol-anglais construit à CMU. La dernière version est Pangloss Mark III avec trois sous-systèmes : PanEBMT (Example-Based MT), PanKBMT (Knowledge-Based MT) et un système de TA avec transfert lexical. Chaque système fournit des résultats avec une convention unique de format. Ces résultats transmis à un module de sélection qui utilise un modèle statistique de langue (Chart). Ce module prend le meilleur « chemin » dans l'ensemble des résultats pour produire un résultat final.

Figure 13: Architecture du système multi-moteur Pangloss



2.1.1.2. VERBMOBIL (2000), des

modules différents mais tous sous Linux

Le système de TA de parole Vermobil [Wahlster, 2000] est destiné à traduire des énoncés oraux spontanés (allemand, anglais, japonais). Une fois la parole reconnue, le format texte est passé en parallèle à 5 modules de traduction. Chaque module suit une approche différente: TA statistique, TA basée sur les cas (case-based MT), TA basée sur les sous-chaînes (substring-based), TA basée sur les actes de dialogue (dialog-act based), et transfert sémantique. Tous ces modules tournent sous Linux. Comme dans PANGLOSS, les résultats sont combinés par un module basé sur un modèle de langue. Une seule sortie est fournie par le système.

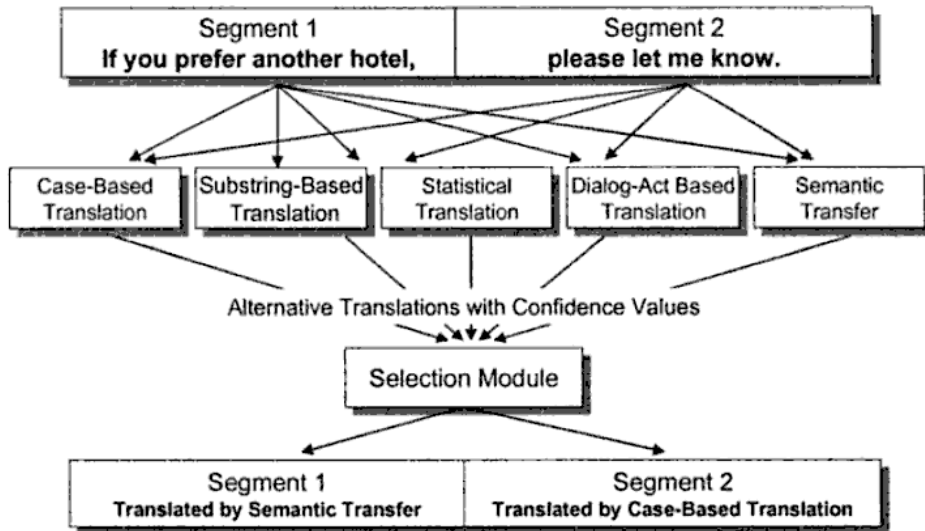


Figure 14: Architecture multi-moteur du système Vermobil

Une autre application des systèmes multiples (multi-moteur) est le module d'identification de la langue d'entrée. Pour chacune des trois langues (allemand, anglais et japonais), il y a un module d'identification de langue séparé. On lance simultanément ces trois modules sur la première seconde de parole de l'entrée et on décide de quelle langue il s'agit par une mesure combinant les trois résultats.

2.1.1.3. ALTFLASH (2001), un système de « TA par patrons » avec un système général en secours

Le système ALTFLASH [Uchino, Ooyama et Furuse, 1999] a été construit pour traduire du japonais en anglais les « brèves » (flash reports) de la bourse de Tokyo (le Nikkei) en temps réel et avec une qualité suffisante pour ne pas avoir besoin de correction avant diffusion.

Le système contient deux sous-systèmes séparés ayant deux approches différentes : le premier est à base de règles et à patrons (bilingues) donnant une traduction directe. Le second est en fait le système ALT-J/E [Ikehara et al., 1991] de NTT, avec des dictionnaires du domaine considéré.

Le premier est utilisé pour traduire des phrases simples (titres, tableaux récapitulatifs) avec patrons fixés. Dans les patrons, on trouve des variables pour les noms d'entreprises, les nombres, etc. Pour une phrase d'entrée, on extrait ces variables, puis on réalise l'analyse morphologique. On traduit alors les mots simples mot à mot par le dictionnaire. Les mots composés sont traduits par ALT-J/E. Le résultat final combine ces deux sorties et passe à la phase de post-édition avec seulement 10% de correction.

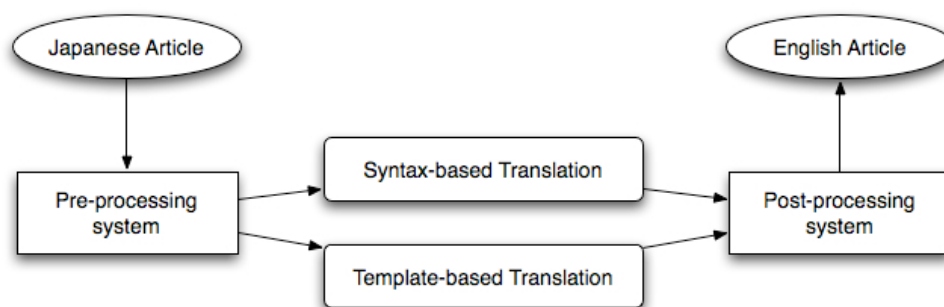


Figure 15: Structure de traduction dans ALTFLASH

2.1.2. Idées en cours, combinaison de résultats de plusieurs systèmes de TA

2.1.2.1. Récupération de résultats de systèmes de TA en ligne

2.1.2.1.1. Soumission et récupération de plusieurs systèmes de TA Web

Pour soumettre et récupérer les résultats de plusieurs systèmes de TA sur le Web, Vo T-H a créé le système TRADOH dans cadre de sa thèse [Vo, 2004a]. Le but de ce méta-système est d'exploiter les traducteurs en ligne disponibles. Pour le faire, il faut simuler la requête à envoyer aux serveurs de traduction, puis récupérer la page de retour et l'analyser pour en extraire les parties d'information concernées (texte traduit).

Un effort similaire est le méta-moteur des systèmes de TA, qui a été réalisé par M. Potet [Potet, 2009] de notre équipe.

Il y a des problèmes difficiles comme la segmentation des documents (pages HTML, textes, ...), l'identification de la langue et du codage des documents ou des zones homogènes dans des documents hétérogènes. Cela est réalisé par le système SANDOH [Vo, 2004b]. D'autres problèmes existent comme, la sélection des traducteurs automatiques, la réutilisation des traducteurs automatiques existants, et la présentation des résultats.

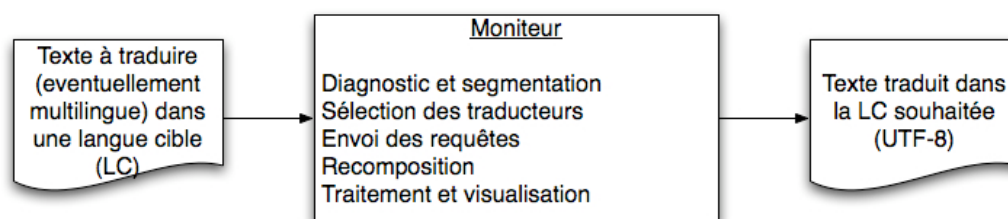


Figure 16: Méta-système TRADOH de traduction automatique multilingue

Dans le cadre de cette thèse, nous avons étendu TRADOH pour qu'il puisse appeler également des traducteurs installés en local sur notre serveur (voir 2.1.2.1.3).

2.1.2.1.2. Utilisation de résultats multiples factorisés

En réalité, dans un contexte de post-édition, on peut avoir plusieurs résultats renvoyés par plusieurs systèmes de TA. Il est cependant difficile de les en lire tous et de sélectionner le meilleur. On se contente donc de proposer les 3 ou 4 premiers, en choisissant un pour initialiser la zone de post-édition. La pratique montre d'ailleurs que les post-éditeurs ne regardent presque jamais les autres propositions de traduction (moins de 5% des cas), et sélectionnent encore moins souvent une autre proposition que la 1% (moins de 0,5% des cas, d'après [Blanchon, Boitet et Huynh, 2009]).

Dans le cas où il ne s'agit pas de post-édition, mais de révision (l'utilisateur ne connaît pas la langue source), il est possible de proposer dans la zone de texte, tandis qu'un panneau montre les possibilités alternatives correspondant à l'endroit « cliqué » du graphe sous-jacent. Il est utile d'ajouter une vue « pidgin multiple » reposant sur un gros dictionnaire bilingue pour indiquer à l'utilisateur les possibilités lexicales (qu'elles soient attestées ou non dans le graphe factorisant les TA).

Par exemple, la traduction d'anglais en français de la phrase « The file is full » obtenue via des systèmes de TA en ligne comme : Google Translate, Systran, Reverso... est :

Système de TA	Résultat pour « The file is full. »
Google Translate	Le dossier est complet.
Systran	Le fichier est complet.
Reverso	Le dossier est plein.
@Prompt	Le dossier est plein.
FreeTranslation	Le dossier est complet.

On peut factoriser les résultats et les représenter sous forme d'un graphe (Figure 17). On peut ajouter ou modifier des nœuds sur ce graphe pour obtenir un meilleur résultat.

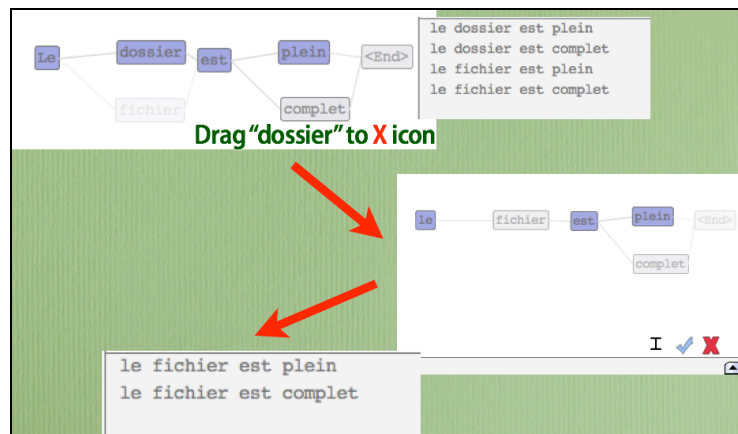


Figure 17: Représentation des résultats de TA par un graphe avec manipulation directe pour sélectionner le meilleur.

2.1.2.1.3. Mise en service en ligne de systèmes de TA locaux

Actuellement, sur les systèmes de TA commerciaux, on peut acheter un produit de type « serveur » qui fournit les services de TA via le réseau. Cependant, ce type de produit est plutôt réservé à de grandes entreprises (riches)⁵. Donc, on voudrait étendre l'accessibilité de ce type de service à des versions moins chères (tournant en local) pour qu'on puisse s'en servir n'importe où. Cela demande d'adapter le programme pour qu'il fournisse au moins un mode d'exécution par ligne de commande, ou par API, sous le système d'exploitation.

À partir de ce service, on peut étendre le système TRADOH [Vo, 2004a] vers TRADOH++, qui peut se connecter non seulement à des systèmes en ligne, mais encore à des systèmes installés sur une machine locale pour répondre à des besoins internes.

⁵ La famille de versions serveur de SYSTRAN coûtait environ de 100 000 à 250 000 € en 2009.

2.1.2.2. Architecture opérationnelle pour les systèmes de TA multiples dans le Web 2.0

2.1.2.2.1. Problème d'organisation dans l'utilisation des contributions publiques de Google Translate

Actuellement, Google fournit un service de traduction automatique en ligne. Sur la traduction, on a la possibilité de corriger le résultat à la volée sur un segment. Cependant, ces modifications ne sont pas effectuées dans la traduction suivante car le système de TA de Google utilise l'approche probabiliste qui se contente de recherche exacte ou approchée dans la MT (beaucoup trop grand). De plus, la traduction n'est pas sensible au domaine il n'y a qu'un seul gros modèle, recalculé de temps en temps).

2.1.2.2.2. iMAG, passerelles d'accès multilingue dédiées

Une iMAG⁶ est une « passerelle de traduction » à première vue très semblable à Systranet⁷ ou à Google Translate⁸. La différence essentielle est qu'elle contient un éditeur de traductions (comme BEYTrans (voir 2.3.2.3), site participatif dédié à des documents et pas à des sites Web), ainsi que des ressources "génériques" (dictionnaires bilingues d'usage en ligne, traductions trouvées sur le Web, corpus parallèles comme EuroParl⁹, appel à des traducteurs automatiques gratuits, support au travail collaboratif de post-édition ou de traduction quand il n'y a pas encore de ressources automatisées pour un couple de langues donné).

En plus de cela, qui suffit à une communauté de traducteurs volontaires comme ceux qui contribuent à la traduction des documents de Pax Humana, d'Amnesty International, *etc.*, à la localisation de logiciels "ouverts" comme Mozilla (voir 2.3.1.1.1), ou de sites dédiés (toursime, support technique). Une iMAG contient *un dictionnaire terminologique et phraséologique spécialisé* au site "élu", et *une mémoire de traductions elle aussi spécialisée* à ce site, et munie de "niveaux de qualité" (de la traduction mot à mot à la traduction certifiée par un traducteur agréé par le site élu). C'est pourquoi on parle de passerelle "dédiée".

Le scénario-type est alors le suivant : si l'on veut accéder à une page Web d'un site élu dans une langue étrangère pour laquelle son iMAG est « équipée », on copie l'URL de cette page dans un formulaire de l'iMAG, on y choisit aussi la langue, et l'on navigue dans le site élu dans la langue choisie. À ce niveau, tout se passe apparemment comme quand on utilise les services linguistiques de Google ou de Systranet.

Il y a cependant deux différences majeures. D'abord, une page en langue "cible" n'est pas construite uniquement par traduction automatique. On la découpe en "segments" (phrases ou paragraphes), et on les traduit par les meilleurs résultats possibles trouvés dans la mémoire, et sinon par traduction automatique. La qualité s'améliorera donc avec le temps.

Ensuite, et c'est l'essentiel, on peut, lors de la lecture d'une page Web du site élu "accédée" en langue cible, sélectionner tout ou partie de la page, et passer en contexte d'édition des traductions et des contributions au dictionnaire. C'est

⁶ Ce texte est extrait de la proposition du projet iMAG/LIG 2008 rédigé par Pr. C.Boitet

⁷ <http://www.systranet.com>

⁸ http://www.google.fr/language_tools?hl=fr

⁹ <http://www.statmt.org/europarl/>

l'aspect wiki de ce concept. Les lecteurs deviennent alors des contributeurs potentiels. Notons que Google Translate propose depuis 2008 aux internautes d'améliorer la qualité des traductions qu'il propose, mais phrase par phrase, et sans aucune spécialisation possible à tel ou tel site.

Deux « maquettes » d'iMAG ont été implémentées, par M.Daoud dans son M2R [Daoud, 2007], et par Carlos Ramisch dans son stage ENSIMAG [Ramisch, 2008]. Dans la seconde, V.Bellynck a intégré un relai de traduction (Redirection Gateway) qui sert d'interface entre l'iMAG et son site élu (en particulier pour la gestion des utilisateurs et des droits).

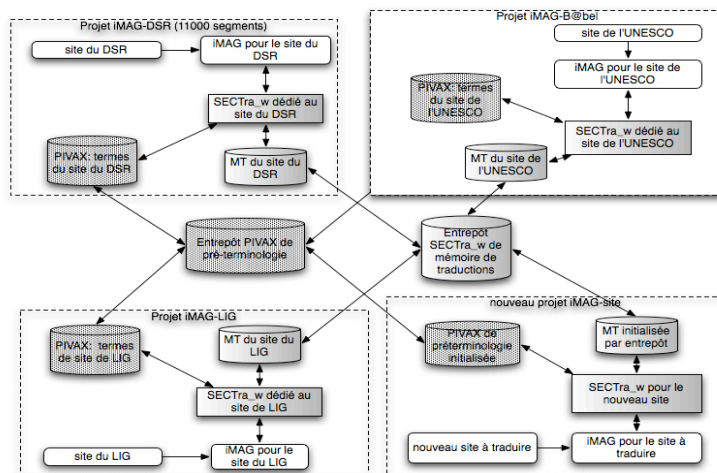
Depuis avril 2009, dans le cadre du contrat Baabel de l'ISCC (CNRS), 7 autres sites iMAG ont été créés. La table suivante donne les URL.

Nom abrégé	Site élu	Date de création
Laboratoire Informatique de Grenoble (LIG)	http://liglab.fr	22/11/2008
Ville de Danang	http://www.danang.gov.vn/	17/07/2008
Tree of Life	http://tolweb.org/	30/09/2009
Unesco/B@bel	www.unesco.org/webworld/babel/	02/05/2009
Digital Silk Road (DSR)	http://dsr.nii.ac.jp/index.html.en	26/04/2009
Centre MICA	http://mica.edu.vn	30/04/2009
ISCC	http://www.iscc.cnrs.fr/	15/05/2009
Site de support technique de SYSTRAN	http://www.systran.fr/support	07/10/2009

Un premier prototype opérationnel a été implémenté par Cong-Phap Huynh dans le cadre du projet iMAG/LIG. Il a permis de démontrer le concept en rendant le site de notre laboratoire LIG¹⁰ accessible dans différentes langues (anglais, allemand, vietnamien...). Pour la segmentation, on a utilisé indirectement le segmenteur de Google Translate.

On envoie la page à traduire à Google Translate, on récupère le résultat, puis on détecte les segments marqués par Google Translate. Dans le relai de traduction, sur chaque segment, on ajoute la possibilité de post-édition et de contribution dans la mémoire de traduction SECTra_w. la page traduite est « reconstruite » à partir du code HTML de la page source, des meilleures traductions des segments (trouvées dans SECTra_w), et de code javascript spécifique.

Le projet iMAG/LIG contient la traduction d'environ environ 2 000 segments sur le site Web de notre laboratoire.



¹⁰ <http://liglab.fr>

Figure 18: Architecture composée de SECTra_w, PIVAX, iMAG

Du point de vue de l'architecture, on construit des instances spécifiques de composants SECTra_w (pour la MT), PIVAX (pour le dictionnaire) et d'une iMAG dédiée. Il y a une connexion entre ces instances locales et une instance centrale de données afin d'initialiser les données pour un nouveau site.

2.2. Systèmes de TA avec processus hétérogènes

2.2.1. Avènement de la TA « fondée sur la connaissance » (1985-1995)

L'idée des systèmes experts a été proposée en 1965 par Edward A. Feigenbaum à l'université de Stanford. Les années 1980 sont considérées comme la période glorieuse des recherches sur les systèmes experts et l'IA, et cela a affecté le domaine de la TA et du TALN.

2.2.1.1. Interaction avec une base de connaissances (thèse Gerber 1984)

Dans sa thèse [Gerber, 1984] au CETA, René Gerber a bien distingué les niveaux de connaissance nécessaires pour « bien traduire » : linguistique (morphologique, logico-sémantique, actualisations...) et extralinguistique (expertise ou pragmatique), ce dernier reposant sur des connaissances approfondies et dépendant du contexte/domaine, ce qui est très difficile à réaliser dans un système de TA. Il a proposé de construire un système de TA expert avec une architecture hétérogène en deux parties :

- un système de TA qui utilise des connaissances linguistiques (par Ariane)
- un système expert qui traite des connaissances extralinguistiques, composé de deux sous-systèmes experts de correction interactive, l'un pour l'analyse et l'autre pour le transfert.

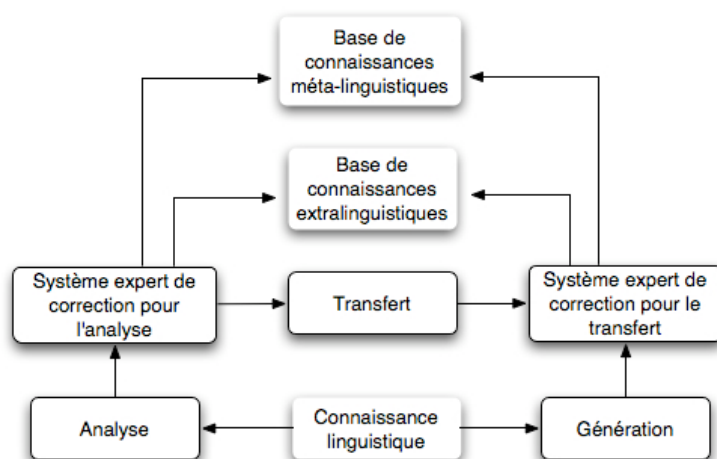


Figure 19: Architecture du prototype de TA experte hétérogène de R. Gerber (1984)

L'architecture est la suivante :

- un système de TA,
- une base de connaissances,
- deux systèmes experts faisant l'interface entre les structures de la langue source produites par le système de TA et la base de connaissance. En analyse, par exemple, le système expert reconnaît des « patrons d'erreur ou de doute » (des schémas de sous-arbre) et les transforme en requêtes de la base de connaissances. Selon la réponse, il modifie ou non l'arbre linguistique, et le renvoie au système de TA.

Le protocole d'échange entre les deux correcteurs et le système de TA écrit en Ariane-G5 est le suivant :

1. Transformer le résultat d'analyse en un format utilisable par le correcteur
2. Tant qu'il y a encore des erreurs à corriger (utiliser la connaissance métalinguistique ou extralinguistique)
 - a. Faire une correction
 - b. Si on ne peut pas faire de correction, terminer
 - c. Reconstruire la sous-partie de structure qui est affectée par la correction
3. Sortir une structure Ariane

L'implémentation a été faite en Foll-PROLOG et testée avec un système prototype anglais-français utilisé pour l'enseignement de la TA.

2.2.1.2. Interaction avec une ontologie dans KBMT-89, puis système KANT-Catalyst

En 1985-89, à CMT, Centre pour la Traduction Automatique à l'université Carnegie Mellon (CMU), S. Nirenburg, J. Carbonnell et leurs collègues ont inventé et prototypé l'approche de TA basée sur la connaissance, ou « KBMT (Knowledge-Based Machine Translation) ». Cette étude a débouché sur un prototype, KBMT-89, appliqué à la traduction de manuels d'installation et de maintenance de PC.

Les ressources utilisées étaient : une ontologie (1 600 concepts avec un modèle du domaine), des dictionnaires d'analyse et de génération (1 200 unités lexicales pour chaque langue) [Mitamura et Nyberg, 1992a], des grammaires syntaxiques, et des règles de correspondance entre les niveaux syntaxique et sémantique.

Le prototypage ayant été jugé positif, CMU, via le groupe Carnegie, a passé un contrat avec Caterpillar pour construire un système opérationnel. Le système KANT¹¹ a d'abord été réalisé comme un prototype opérationnel pour la recherche et le développement de systèmes de traduction à grande échelle pour la documentation technique. Ce système utilisait des vocabulaires et des grammaires spécifiques pour chaque langue source, et des modèles sémantiques explicites pour arriver à une traduction de haute qualité dans chaque domaine technique.

KANT a ensuite été spécialisé pour donner d'autres systèmes dédiés à des sous-langages: production de documentation multilingue dans les domaines de l'électroménager (projet ESTRATO), documentation de machines lourdes (projet CATALYST), manuels de voitures, de téléviseurs... Après 3 ans, ce système avait environ 14 000 sens de mots généraux (correspondants à moins de 5 000 termes) et des centaines de termes dans des domaines spécifiques.

Le principe de KANT [Mitamura et Nyberg, 1992b] est d'utiliser une architecture linguistique à « pivot sémantique » (IF, Interlingua Frame).

Cela permet d'ajouter une nouvelle langue avec une équipe de développement et une représentation linguistique de langue Li indépendamment des autres langues. Par contre, cela demande des connaissances approfondies au niveau du langage pivot afin de ne pas perdre l'information à l'étape de génération pour toutes les langues.

¹¹ Projet KANT, <http://www.lti.cs.cmu.edu/Research/Kant/>, visité en octobre 2008

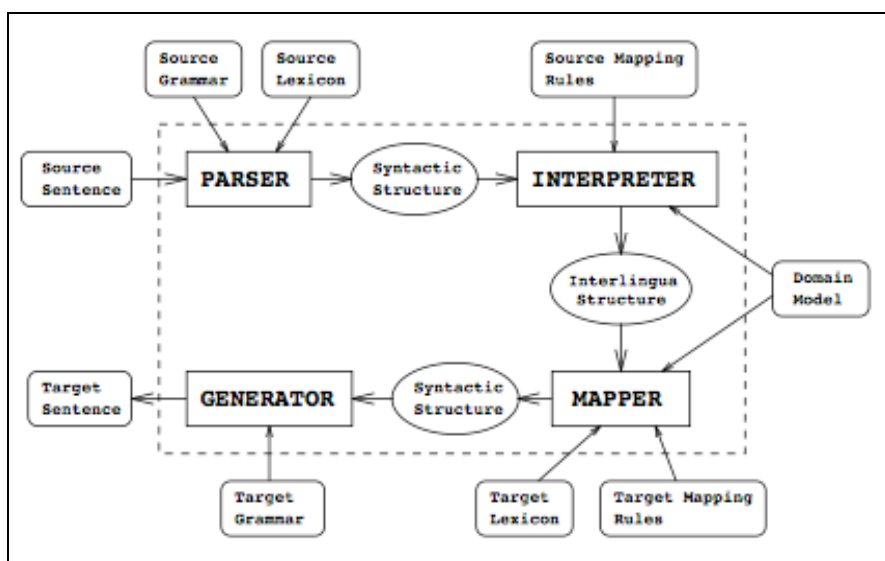


Figure 20: Architecture du système KANT

D'autre part, au cours de la traduction, le système KANT permet aux utilisateurs deux modes de désambiguïsation : interactive (demander aux utilisateurs de choisir et éventuellement annoter des entrées en SGML) et automatique (utiliser le modèle de domaine pour désambiguïser et des heuristiques pour choisir en fonction de préférences exprimées sous forme de règle). Voici quelques exemples de désambiguïsation dans KANTOO :

- Choisir la préférence

"The man saw the boy with the telescope."
 "The man saw the boy with the dog."

Ces deux cas sont syntaxiquement corrects, mais l'information de préférence décide lequel est choisi.

- Modèle sémantique du domaine

Le modèle sémantique du domaine est un triplet : (<head> <semantic-role> <filler>).

(*A-LIFT INSTRUMENT *O-HOIST)

"Lift the engine with a hoist."

- Sens de mot

« Turn the truck to the right. »	« The deposits turn into sludge. »
A-TURN-1: "demander de tourner"	A-TURN-2: "changer le statut ou forme"
(*A-TURN-1 (ORIENTED TO *O-RIGHT-1))	(*A-TURN-2 (RESULT INTO *O-SLUDGE))
<p>(*A-TURN-1 (argument-class agent+theme) (mood imperative) (punctuation period) (tense present) (q-modifier (*Q-oriented_TO (case (*K-TO)) (object (*O-RIGHT-1 (number (:OR mass singular)) (reference definite))) (role oriented))) (theme (*O-TRUCK (number singular (reference definite))))))</p>	<p>(*A-TURN-2 (argument-class theme) (mood declarative) (punctuation period) (tense present) (q-modifier (*Q-result_INTRO (case (*K-INTO)) (object (*O-SLUDGE (number mass) (reference no-reference))) (role result))) (theme (*O-DEPOSIT (number plural (reference definite))))))</p>
Haga girar el camión a la derecha.	Los depósitos se convierten en sedimento
GLOSS: MAKE TURN THE TRUCK TO THE RIGHT	GLOSS: THE DEPOSITS ARE TURNED INTO SLUDGE

La nouvelle version commerciale de KANT, implémentée en C++, est nommée KANTOO [Mitamura et Nyberg, 2000]. Le framekit KANTOO a été complètement réimplémenté à partir de celui de KANT en ajoutant des composants de support de création et de mise à jour des termes (Lexical Maintenance Tool, LMT) et des connaissances (Knowledge Maintenance Tool, KMT) pour des applications de TA dans KANTOO.

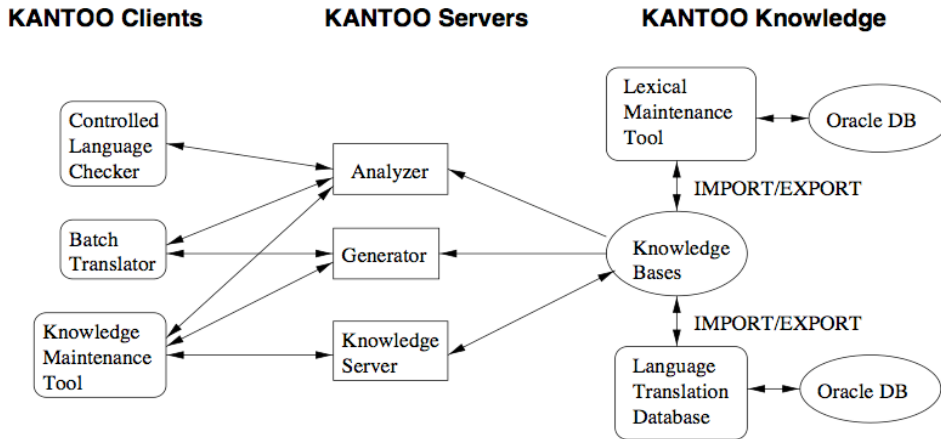


Figure 21: Architecture du système KANTOO

Le module LMT est utilisé par tous les utilisateurs, mais le module KMT est spécialement réservé aux experts ou linguistes dans un cycle de développement d'une application de TA.

2.2.1.3. Interaction entre la connaissance et le modèle statistique pour traduction de parole, système MASTOR (IBM)

Le système MASTOR (Multilingual Automatic Speech-to-Speech Translator) est développé depuis 2001 au centre de recherche d'IBM pour traduire de la parole spontanée en bidirection anglais-mandarin [Gao et al., 2006]. L'architecture du système est la suivante :

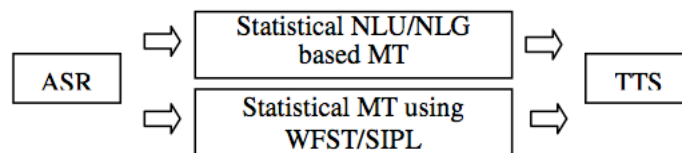


Figure 22: Système de traduction de parole MASTOR d'IBM

Après avoir passé le module de reconnaissance de parole (ASR), la phrase est traduite en parallèle par deux systèmes, l'un probabiliste (Statistical NLU/NLG), l'autre utilisant des automates (WFST/SIPL).

Une traduction statistique typique (voir 1.2.3.2.1.1) de la langue source W à la langue cible A est paramétrée par un modèle $p(A|W)$ entraîné à partir d'un corpus bilingue. Cependant, dans le système MASTOR, on a annoté les phrases du corpus par des concepts pour que le choix optimal d'une phrase cible A pour une phrase source W donnée ne soit pas fondé que sur l'occurrence des A et W mais encore sur le sens associé. Si S dénote la représentation conceptuelle de la phrase source, C celle de la phrase cible. Donc, l'algorithme de sélection une meilleure séquence de mots est :

$$\hat{A} = \arg \max_A p(A|W) = \arg \max_A \left\{ \sum_{S,C} p(A|S,C,W) p(S|C,W) p(C|W) \right\}$$

$p(C|W)$, $p(S|C,W)$, $p(A|S,C,W)$ sont estimés par le module de reconnaissance de langue (NLU, Natural Language Understanding), le module de génération conceptuelle (NCG,

Natural Concept Generation), et le module de génération morphologique (NWG, Natural Word Generation).

Le système MASTOR fonctionne en temps réel sur un PC ou un PDA (256Mh, 64 Mo de RAM) avec la traduction d'environ 30K mots sur certains domaines.

2.2.2.Interaction avec des humains

Un système de TA peut aussi aller « chercher la connaissance » dans la tête des utilisateurs humains ; on parle alors de « TA interactive » et de « désambiguïsation interactive ».

On voit dans cette partie des stratégies différentes dans la réalisation de cette fonction, soit en phase d'analyse et de transfert (ITS-1), soit entre les phases (projet LIDIA), soit les deux, en retardant le plus possible, comme dans le système ITS-2.

2.2.2.1.Interaction en analyse et transfert à ITS-1 (Eldon G.Lytle, BYU, Provo, 1972-1981)

Le système ITS-1 (Interactive Translation System) a été développé par une équipe du TSI (Translation Service Institute) à BYU, dirigée par Eldon G. Lytle, à laquelle A. K. Melby [Melby, Smith et Peterson, 1980] participe pendant la période 1972-1981. La phase d'analyse est indépendante de langue cible et la phase de génération est quasiment indépendante de la langue source. Le texte source d'entrée du système a en général plusieurs auteurs et styles différents.

On a expérimenté deux modes de traduction. Le premier a utilisé un éditeur avec un support de dictionnaire par téléphone pour taper la traduction. La deuxième part du fichier d'entrée sur une station de travail en ligne de commandes (le PC avec GUI n'existait pas encore), on fait désambiguïser interactivement par l'humain (cela prenait 10 minutes par page) sous forme de questions, et le système traduit en langue cible. Le résultat du deuxième mode était nettement meilleur que celui du premier.

Dans le système ITS-1, on était bien conscient que le coût d'interaction humaine était relativement élevé. Donc, on a d'abord utilisé la désambiguïsation et la correction humaine seulement dans la phase d'analyse de la langue source (anglais). Ce résultat était envoyé à plusieurs modules de transfert et de génération de toutes les langues cibles (espagnol, portugais, allemand, français et chinois (très basique)). Cependant, l'expérience a montré qu'il fallait intervenir également sur le transfert et la génération pour toutes les langues cible. Pour traduire une page standard (250 mots), il fallait 7 minutes sur la phase d'analyse, 10 minutes sur la phase de transfert et 10 minutes de post-édition. Le temps total de 27 minutes était un peu inférieur à celui de la traduction par un professionnel utilisant une bonne mémoire de traduction. Cependant, cette différence n'était pas justifiable pour la contribution lexicale de plus dans le processus de traduction dans l'environnement d'ITS-1.

2.2.2.2.Interaction le plus tard possible dans ITS-2 (Éric Wehrli, Genève, LATL)

Le système de TA ITS-2 (Interactive Translation System¹²) [Wehrli et Ramluckun, 1993] développé au LATL (Laboratoire d'Analyse et de Technologie du Langage) de l'université de Genève tourne sous Windows et opère sur les langues : anglais, français, allemand et italien.

ITS-2 se base sur l'approche de transfert, et l'étape d'analyse est le module IPS (Interactive Parsing System) [Wehrli, 1992]. L'interaction se fait par des dialogues de clarification à plusieurs niveaux :

- Niveau lexicographique : quand le texte d'entrée contient des mots non connus ou erronés, on demande aux utilisateurs de les corriger,

¹² ITS-2 est indépendant du système ITS-1 de BYU

- Niveau syntaxique : quand on a des ambiguïtés non résolues par l'analyseur nécessitant des connaissances extra-linguistique ou contextuelles, on demande à l'utilisateur de choisir,
- Niveau de transfert : le système demande aux utilisateurs de sélectionner des équivalents en langue cible.

Dans le projet courant (ITS-3), on a voulu simplifier les interactions en se limitant au niveau du transfert lexical. Le principe du système ITS-3 est qu'on voudrait faire le moins d'intervention humaine possible. La machine essaie d'abord de résoudre l'ambiguïté ou la laisse pour l'étape suivante (pour les cas d'ambiguïté en cours d'analyse, le système conserve les possibilités et propose un choix dans l'étape de transfert si possible). Quand la machine ne peut pas résoudre l'ambiguïté, elle propose à l'utilisateur la suite des possibilités depuis la première phase. L'utilisateur est passif dans ce contexte.

Le système fonctionne sous Windows avec un éditeur intégré. La désambiguïstation est faite par choix des catégories proposées dans des menus contextuels sur la phase en question. Il y a également un prototype d'ITS-3 sur le Web [L'haire, Mengon et Laenzlinger, 2000].

2.2.2.3. Interaction entre 2 phases dans le projet LIDIA (1989-1996)

L'approche de « TA fondée sur le dialogue avec l'auteur » (DBMT, Dialog-Based Machine Translation) a été définie et expérimentée dans le projet LIDIA (Large Internationalisation des Documents par Interaction avec leurs Auteurs) du GETA, à Grenoble, en 1989-1996 [Blanchon et Boitet, 1994]. La première maquette a été implémentée dans un système hétérogène intégrant un « serveur de TA » écrit en Ariane-G5 et un « désambiguïseur interactif » écrit en Common Lisp et Hypercard sur Macintosh. Le scénario était la traduction de piles Hypercard pour la production de documents techniques multilingues (traduction de français vers allemand, russe et anglais).

Le principe de LIDIA est que l'utilisateur peut intervenir pour faire la désambiguïstation interactive en choisissant les solutions sur un arbre de questions proposées sous forme de dialogues « naïfs » (Figure 23). Ces interventions ne sont pas faites durant une phase mais sur une représentation intermédiaire, à un seul endroit.

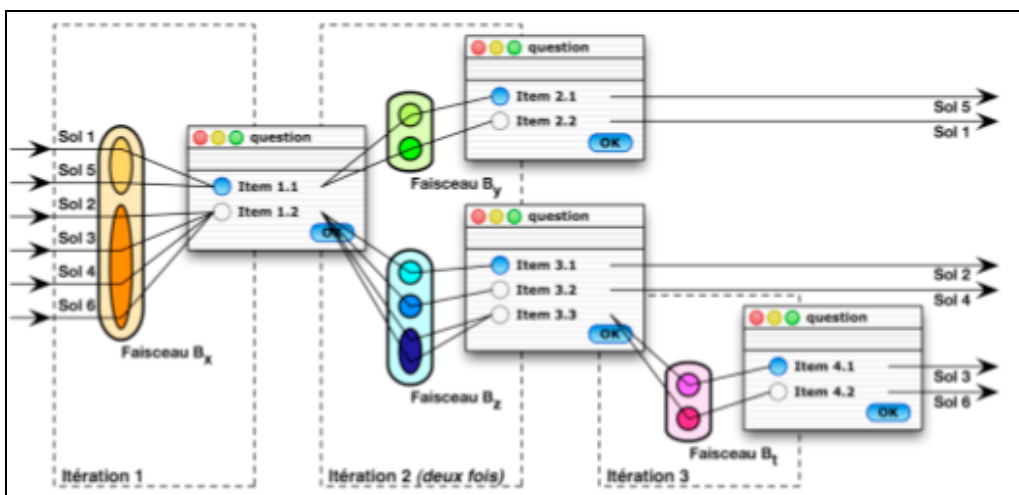


Figure 23: Construction d'un arbre de question en vue externe de LIDIA

Pour ce faire, dans la maquette LIDIA-1, on a utilisé une architecture logicielle distribuée avec trois modules : le module de rédaction basé sur HyperCard sur machine locale (Macintosh), le module de traduction qui fait totalement appel à Ariane-G5 via le réseau en

utilisant le protocole SMTP (courriel)¹³, et enfin le module de désambiguïsation. Ils communiquent par AppleEvents, à travers des interfaces de communication : client de rédaction, serveur de désambiguïsation, serveur de communication. L'ordonnancement des traitements est assuré par un coordinateur.

Sur un texte d'entrée, on utilisait des transcritteurs de caractères (certains en LT, voir 3.2.3.3.2) pour harmoniser les codages différents entre Mac et Ariane-G5. On envoie chaque phrase au système de TA écrit en Ariane-G5 pour qu'il l'analyse. Le résultat de la phase d'analyse est multiple. Ces arbres sont envoyés au module de désambiguïsation sur lequel on a construit des compilateurs d'arbres, qui transforment les deux formats d'arbre produits par Ariane-G5 du/au format LIDIA_Lisp. Sur le Mac, on a un LSPL implémenté en Common Lisp (MCL) pour décrire des règles de résolution interactive d'ambiguïtés. À partir de ces règles et des données au format LIDIA_Lisp, on génère du code HyperTalk pour produire l'interface de dialogue. L'utilisateur sélectionne une suite de réponses dans des menus présentés par cette interface et peut demander éventuellement de faire un retraduction vers la langue source. Après avoir fini la désambiguïsation, les données sont recompilées dans le format d'Ariane-G5 et envoyées à la phase de génération pour avoir le texte en langue cible.

Le bilan critique (extrait de [Blanchon et Boitet, 2004]) de l'architecture informatique des maquettes LIDIA-1 effectué en 2000 a fait apparaître différents problèmes, d'où la réalisation en 2001 de LIDIA-2, une version en Java de l'environnement d'accès aux services de TAFD. L'environnement d'accès aux services LIDIA est un micro-éditeur de documents XML très limité. Le document produit contient le texte rédigé par l'auteur et l'historique de la désambiguïsation interactive. Dans cette version, H. Blanchon a utilisé un désambiguïseur de l'anglais directement utilisable dans la nouvelle architecture (Figure 24).

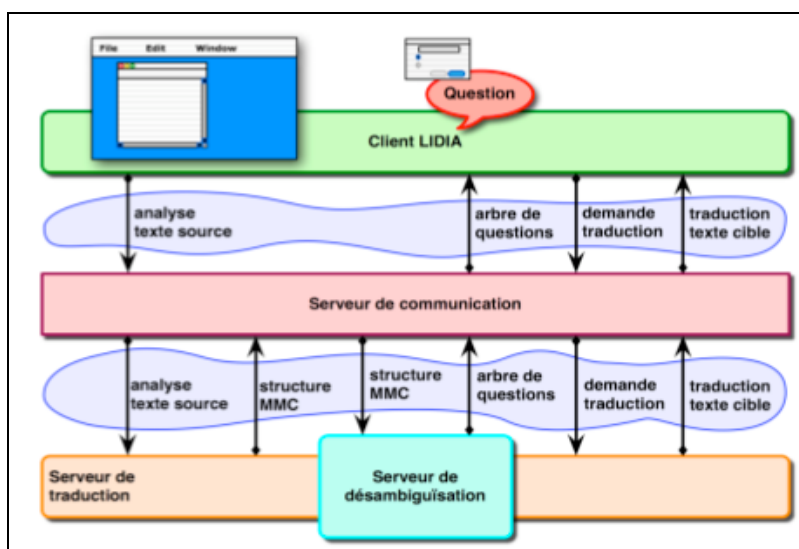


Figure 24: Architecture logicielle mise en oeuvre en LIDIA-2

2.2.3. Architectures computationnelles et EDL différents

Vu le coût de construction de systèmes de TA, il est naturel de chercher à en construire de nouveaux en utilisant des composants linguiciels provenant de plusieurs systèmes. Soit on programme des transferts entre structures associées à 2 langues naturelles L1 et L2, soit on passe par un langage pivot « interlingue » plus ou moins abstrait.

¹³ Les versions ultérieures ont utilisé une communication par sockets.

Parmi les projets et systèmes existants, nous citons ici quelques exemples opérationnels, par diversité croissante: maquette de combinaison MU-Ariane, projet MMT du CICC, et projet UNL en cours.

2.2.3.1. Maquette MU → FR (1984-1985)

Une autre essai de combinaison de composants est la maquette de TA japonais-français réalisée en 1984-1985 avec le système Ariane-G5 (GETA, Grenoble) et le système MU (Université de Kyoto, Japon) avec la participation de J. Koca (Ensimag, Grenoble) et S. Sato (NEC). Un texte à traduire en japonais est analysé par le système MU, puis le résultat arborescent de l'étape d'analyse est envoyé à Ariane. Un transfert « maquette » a été réalisé pour cela. Ensuite, on « branche » la sortie de l'analyse sur l'entrée de la génération du français du système russe-français (RUB-FRB).

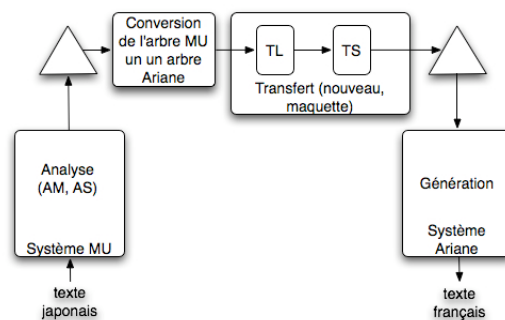


Figure 25: Maquette de système de TA hétérogène japonais-français combinant MU et Ariane/RUB-FRB [Koca & Sato, 1985]

2.2.3.2. Le projet MMT du CICC (1987-1993)

Le projet MMT (Multi-Lingual Machine Translation) [Funaki, 1993] du CICC¹⁴ est un projet financé par l'ODA¹⁵ pour développer un système de TAO « à pivot » entre 5 langues asiatiques : japonais, chinois, thaï, malais et indonésien. cinq grosses sociétés japonaises actives en TAO¹⁶ ont y participé. On souhaitait obtenir un système de haute qualité (80-90% sans modification) et de haute performance (5 000 mots/h).

Pour chaque langue naturelle à traduire, on construit un module d'analyse morpho-syntaxique et sémantique et un module de transfert de LN (Langue Naturelle) vers IL (Interlingua). Le dictionnaire d'IL contient des descriptions détaillées du transfert lexical des « concepts » et attributs (sémantique) de l'IL, permettant de traduire vers toutes les langues cibles. Le format de dictionnaire est défini par le partenaire EDR (Japan Electronics Dictionary Research Institute, Ltd). Pour une nouvelle langue cible à ajouter, on ajoute la description d'IL vers cette langue dans le dictionnaire, et on développe un module de génération qui peut être indépendant et tourner sur un autre système.

Donc, avec cette approche, on a la possibilité de coopérer de façon assez indépendante avec d'autres centres étrangers. Ces partenaires ont été chinois (China National Computer Software and Technology Service Corporation, CS&S), indonésien (Agency for the Assessment and Application of Technology, BPPT), malaisien (Ministry of Education, MOE), thaï (National Electronics and Computer Technology Center, NECTEC) et japonais (EDR, Electronics Dictionary Research Institute, Ltd).

¹⁴ Center of International Cooperation for Computerization, <http://www.cicc.or.jp>

¹⁵ Overseas Development Agency.

¹⁶ Fujitsu (ATLAS), NEC (PIVOT), Hitachi (HICATS), Toshiba(ATN), EDR Ltd.

2.2.3.3. Le projet UNL (1996-)

Le projet UNL (Universal Networking Language) a commencé en 1996 à l'IAS (Institute of Advanced Studies) de l'Université des Nations Unies (UNU). Son concepteur et directeur est Hiroshi Uchida, le créateur d'ATLAS-II, qui avait quitté Fujitsu fin 1993 après le projet MMT (voir 2.2.3.1). La motivation de ce projet est que le développement d'Internet va faciliter la transmission de l'information, mais va aussi aggraver le déséquilibre entre ceux qui ont accès au réseau et les autres.

Pour arriver à une diffusion et à une communication largement multilingue, il faut, d'après H. Uchida, avoir un système pour exprimer la connaissance sur Internet, et traduire cette connaissance rapidement vers les autres langues naturelles.

Il ne s'agit pas de construire un système de TA, mais plutôt un système de communication multilingue à large spectre (documentation technique, aussi bien que messages personnels ou informations générales et recherche d'information).

L'organisation se compose de deux parties: le centre UNL et les centres de langues. Le centre UNL se situe à Tokyo et à Genève et s'occupe de publier les spécifications, de définir la base de connaissances (KB, Knowledge Base), et de l'administration. Les centres locaux s'occupent du développement des modules concernant leur langue.

En 2001, la fondation UNDL¹⁷ a été établie à Genève pour la gestion financière et administrative et le centre UNL continue à s'occuper des aspects techniques.

Le projet UNL a été lancé avec un plan sur dix ans. Pour la phrase 1 (1997-1999), la tâche principale a été d'établir les spécifications du langage UNL, de développer les «décodeurs» (modules de transformation entre UNL et les langues naturelles), les dictionnaires, la « base de connaissances », et aussi les outils Web.

Après ces trois ans, il y avait 16 partenaires, avec plus de 13 langues (anglais, arabe, chinois, espagnol, français, hindi, indonésien, italien, japonais, mongol, portugais, russe et thaï) ayant leur «décodeur» depuis UNL, développés avec leurs propres outils, ou bien en utilisant le langage spécialisé Deco proposé par le centre UNL.

¹⁷ UNDL, <http://www.undl.org/>, visité en novembre 2008

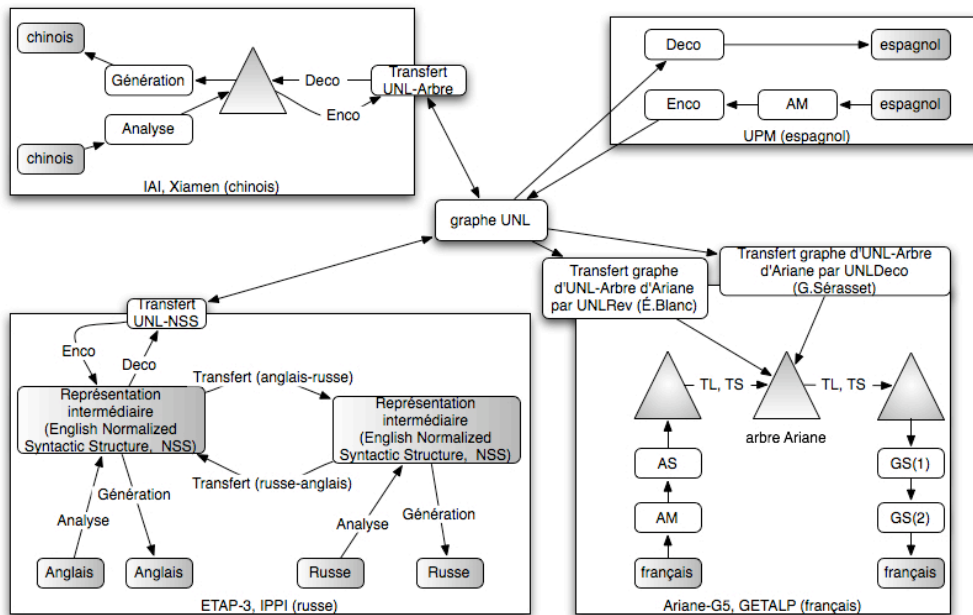


Figure 26: Architecture du système UNL

Le pivot UNL est un interlingua dont les « énoncés » sont des hypergraphes « linguistico-sémantiques ». Chaque nœud porte un UW (« Universal Word » ou acception interlingue) et des attributs sémantiques. Voici l'exemple d'un graphe UNL pour la phrase : « *Le chat attrape une souris et la mange.* »

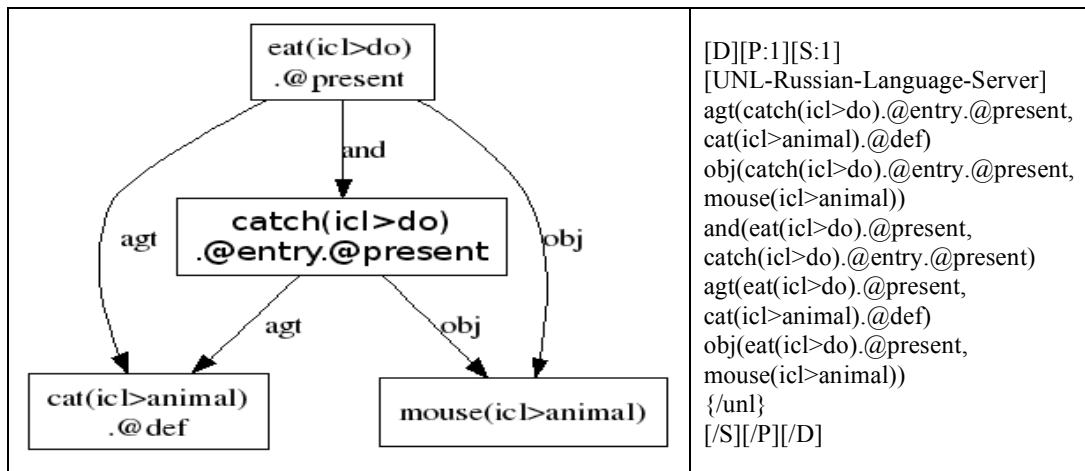


Figure 27: Exemple d'un graphe d'UNL

Une des différences entre le projet UNL et les projets de TA avec pivot interlingue est qu'UNL est spécialement conçu pour l'environnement Internet. H. Uchida affiche même l'ambition d'utiliser le langage UNL comme représentation intermédiaire de la connaissance sur Internet. L'architecture logicielle du système est très hétérogène au sens où chaque centre de langue développe indépendamment son déconvertisseur et son enconvertisseur, et où l'échange de graphes UNL est fait par la connexion Internet.

La réalisation du module de déconversion par chaque centre est très hétérogène : le centre de langue du russe (IPPI, Moscou) utilise son système ETAP-3¹⁸, le centre de langue espagnol¹⁹

¹⁸ UNL russian Language Server, <http://www.unl.ru/>, visité en novembre 2008.

(UPM, Madrid) utilise les langages Deco et Enco du centre UNL, et le centre de langue français (GETALP, LIG, Grenoble) utilise Ariane-G5.

Pour créer un déconvertisseur du français, on pourrait utiliser le LSPL Deco proposé par le centre UNL. Cependant, il est évidemment préférable de réutiliser les composants existants pour la génération du français développés en Ariane-G5. Une autre raison de ne pas utiliser Deco est qu'il n'est pas bien adapté à la génération morphologique (1 000 règles pour l'italien, 10 000 règles pour le russe, tandis que la génération du français²⁰ développée en Ariane-G5 ne contient que 20 règles et 350 affixes sous SYGMOR).

En utilisant cette approche hétérogène, le déconvertisseur UNL-FR est constitué de 7 phases (Figure 28). Les trois premières phases sont la transformation d'un graphe UNL vers un arbre décoré prêt à être envoyé à la phase TL dans Ariane. On a implémenté ces trois phases en deux versions : l'une par Gilles Sérasset sur le Web [Sérasset et Boitet, 1999], l'autre par Étienne Blanc en HyperCard [Blanc, 2000], puis en Revolution²¹.

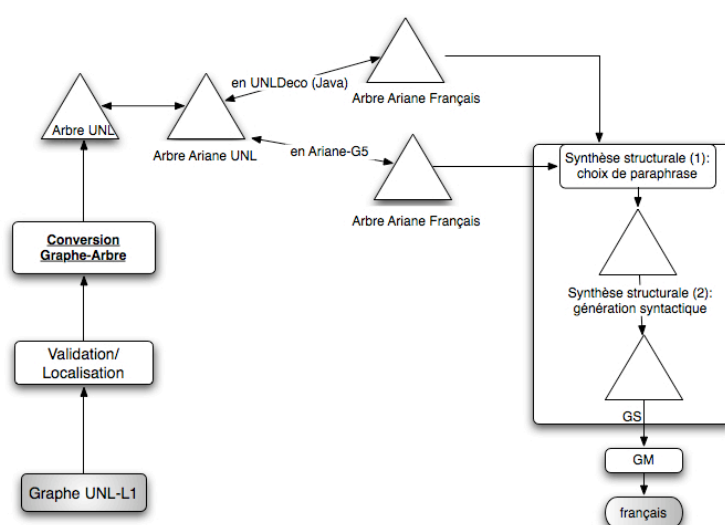


Figure 28: Architecture du déconvertisseur du français du GETALP

2.3. Vers des systèmes de TA et de THAM intégrés & collaboratifs

La croissance importante de l'utilisation d'Internet et des technologies sur le Web affectent les méthodes et les environnements de traduction. Il s'agit d'une classe de systèmes collaboratifs.

On va voir cet aspect dans les systèmes de TH, puis de TH avec appel de TA intégré.

2.3.1. Systèmes de TH collaboratifs

L'idée des systèmes de TH collaboratifs est de faire contribuer à la réalisation des ressources une communauté ou un groupe de personnes grâce à un environnement convivial.

La collaboration est naturelle dans des applications comme la localisation des logiciels libres, la traduction bénévole à la Wiki ou la traduction pour des « causes ».

¹⁹ UNL spanish Language Center, <http://www.unl.fi.upm.es/english/index.htm>, visité en novembre 2008.

²⁰ Cette génération est exhaustive et traite aussi bien les aspects morphologiques que typographiques.

²¹ <http://www.runrev.com/>, visité en juin 2009.

2.3.1.1. Localisation interne de logiciels libres

2.3.1.1.1. MLP, un projet de localisation de logiciels

Dans les logiciels ouverts de la suite Mozilla, on peut localiser l'interface des logiciels ainsi que les dictionnaires des vérificateurs d'orthographe intégrés. Le projet MLP²² (Mozilla Localization Project) fournit aux utilisateurs des moyens et outils pour étendre ou éditer eux-mêmes l'interface via les fichiers de ressources externes, qui ont un format connu et défini par Mozilla.

Comme ces fichiers source font partie du code source dans un projet géré par CVS, chacun fait un checkout et contribue directement sur les fichiers.

Il y a un outil (MozExp Tool) pour extraire les messages de la version installée, puis afficher ces messages dans l'interface à traduire (Mozilla Translator). Après avoir fini, on doit recompiler et reconstruire (rebuild) l'application sur sa machine locale.

Pour publier un nouveau paquet de langues, il faut envoyer les fichiers de ressources pour qu'ils soient révisés par un spécialiste agréé. Pour la terminologie, chaque utilisateur a un fichier de glossaire séparé. On peut télécharger les fichiers de ressources d'interface et les dictionnaires de vérification d'orthographe disponibles sur les langues via le site de MLP.

2.3.1.1.2. Launchpad Translation, un outil de gestion de projets de localisation logicielle

Launchpad Translation²³ est un service de gestion de la traduction des messages et des interfaces pour des projets en source ouvert. Il y a une communauté de 400 000 bénévoles avec 272 langues de localisation. Il y a des projets célèbres qui sont hébergés sur Launchpad, comme Ubuntu, et MySQL.

Il y a deux modes de contribution : soit en ligne, et alors un bénévole édite directement les messages par une interface Web de type Wiki, soit hors ligne, et alors on télécharge et on édite des paquets de ressources.

Dans le deuxième mode, si le logiciel utilise le format standard de ressources (PO, MO), on peut changer directement les messages et l'interface pendant que logiciel tourne. Cela permet d'assurer la cohérence du contexte de traduction. Quand on a fini, on peut synchroniser les ressources contribuées en local avec la version hébergée sur Launchpad. Les contributions sont révisées par un groupe de linguistes (défini selon le projet) avant de les intégrer dans la version suivante du logiciel.

2.3.1.1.3. Traduction de Facebook par ses utilisateurs

L'interface de réseau social Facebook²⁴ peut être traduite et/ou révisée par l'utilisateur dans sa langue. La communauté actuelle réunit 30 000 contributeurs participant aux traductions de l'interface de Facebook dans 63 langues.

Facebook offre un environnement de traduction en ligne (Facebook Translation²⁵). L'utilisateur peut naviguer sur des messages et voter pour une traduction parmi plusieurs propositions ou les traduire dans sa langue préférée. Il y a un glossaire des mots pour un message traduit. La traduction choisie (par un système de votes) est immédiatement mise en place dans la langue traduite sans révision par des traducteurs professionnels.

²² <http://www-archive.mozilla.org/projects/l10n/>

²³ <https://translations.launchpad.net/>

²⁴ <http://facebook.com>

²⁵ <http://www.facebook.com/translations/>

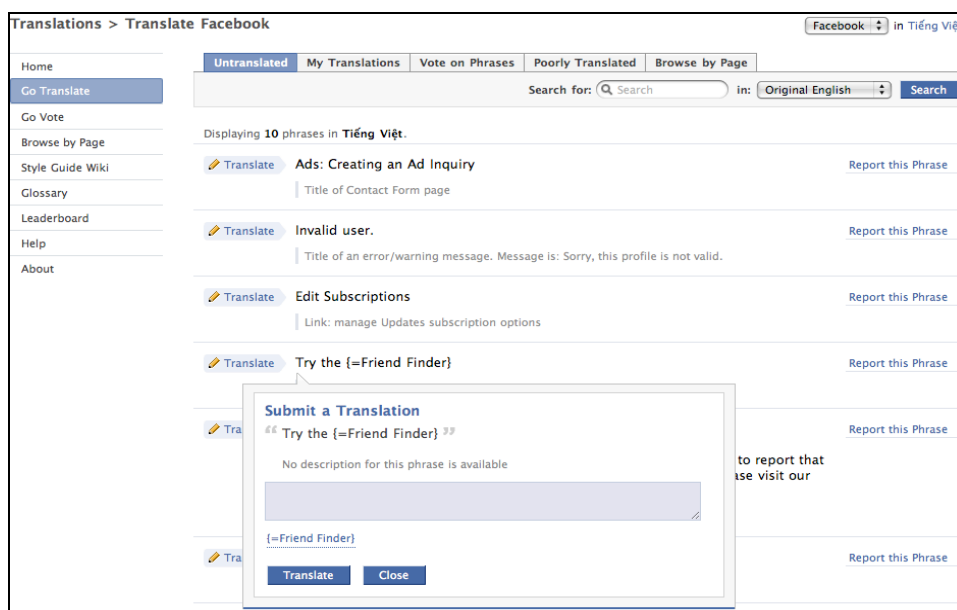


Figure 29: Interface de l'environnement de traduction de l'interface de Facebook

Caractéristiques communes

Les caractéristiques communes de ces projets sont :

- Terminologie : une liste des termes existe au début, mais elle n'est pas gérée de façon contributive, n'évolue donc pas, et est peu ou pas utilisée.
- Traduire nécessite des compétences étroitement liées à l'information.
- Un fichier à traduire est traité par un seul traducteur à la fois.

2.3.1.2. Wikisation, communautés de contributeurs bénévoles

Suite au succès de Wikipedia et à la croissance des communautés de bénévoles, on trouve de nombreuses variantes adaptées aux divers environnements de traduction, comme :

- Pour la traduction des logiciels/systèmes : GNUUnited Nations, TraduWiki, World Wide Lexicon, OLPC wiki multi-lingual hack, Nooku, Der Mundo.
- Pour les ressources de traduction : OmegaWiki.
- Pour la MT : My Memory, VLTM: Very Large Translation Memory, Open-Tran.eu
- Pour la localisation : BetaWiki, TWiki Localisation.
- Pour les projets de traduction : traduction de Wikipédia, TWiki Xchange (échanger les contenus entre les instances d'installation de TWiki et les organiser par un flot de travaux de traduction).
- Pour les conférences : Wiki Symposium, Open Translation Tools 2007 Wiki, Localization World conference...

Parmi ces exemples, on décrit ci-dessous TraduWiki (outil d'aide à la traduction d'un site Wiki) et Pootle (un portail de gestion des traductions à la Wiki).

2.3.1.2.1. TraduWiki

TraduWiki²⁶ est un environnement de traduction collaborative grâce auquel des bénévoles peuvent contribuer à la traduction de livres, de blogs, de sites Web... dans le cadre d'un groupe de personnes ou d'une communauté.

²⁶ <http://traduwiki.org/>

Après s'être inscrit, on peut librement ajouter la traduction d'une langue à un document existant ou créer un document. On peut visualiser l'état des traductions dans plusieurs langues.

2.3.1.2.2. Pootle, un portail de gestion de traductions en ligne

Pootle²⁷ est un portail de traduction en ligne, qui facilite le processus de traduction avec les fonctions suivantes : traduction en ligne, gestion de flux de travaux de traduction, statistiques sur le statut des traductions, contribution des bénévoles...

Pootle est un centre de stockage des documents à traduire. On peut travailler en local avec un ensemble d'outils (Translate Toolkit) associés à Pootle. La connexion entre outils en local et un serveur déclaré est établie par des API.

Caractéristiques communes

Les caractéristiques communes de ces projets sont :

- La plupart de ces outils n'a pas de dictionnaire intégré, ni MT, ni TA,
- On n'a pas besoin de compétence technique,
- On ne peut traduire qu'un seul morceau de texte (segment) à la fois.

2.3.1.3. Support linguistique pour des projets de traduction pour des « causes »

2.3.1.3.1. QRLex

Dans le cadre de sa thèse [Bey, 2008], Y. Bey a contribué à QRLex [Bey, Kageura et Boitet, 2006], une plate-forme fournissant des ressources et des outils aux traducteurs bénévoles travaillant sur leur PC avec communication de ressources et outils accessibles sur Internet. Ces ressources et outils incluent la gestion des documents traduits, et des supports au processus de traduction : consultation de dictionnaires, proposition de « suggestions » de traduction pour les segments déjà traduits dans des documents existant dans le système et sur le Web, intégration de fonctionnalités de TA.

2.3.1.3.2. Projet Pax Humana, Amnesty International

Depuis 1984, Pax Humana [Humana, 2009] a une communauté de plus de soixante traducteurs bénévoles pour traduire des articles dans domaine de la paix et des Droits de l'Homme.

Ces bénévoles sont situés dans plusieurs pays (France, Belgique, Suisse, Canada, États-Unis, Japon, Mexique, Argentine...). Chacun traduit ponctuellement ou régulièrement des textes selon le temps libre dont il dispose. Les traducteurs travaillent souvent en binôme avec un autre collaborateur. Après avoir fini, on dépose en ligne le document résultat pour que les autres puissent le réviser.

2.3.1.3.3. Récapitulatif

Projet	Unité de traduction	Collaboration en ligne	Dictionnaire/ Terminologie	Appel de TA	MT intégré
<i>MLP</i>	Fichier		x (local)		
<i>Launchpad</i>	Fichier		x		
<i>Wiki-translation</i>	Segment	x			
<i>Translation Wiki</i>	Segment				
<i>QRLex</i>	Document	x	x		
<i>Pax Humana</i>	Document	x			

²⁷ <http://translate.sourceforge.net/wiki/pootle/>

2.3.2.Systèmes de TH intégrant l'appel à des systèmes de TA

2.3.2.1.TransActive (ALPS, 1983-1990)

Le système de TAO interactif TransActive d'ALPS (voir 1.2.1.2) avait été créé par des membres de l'équipe de A. Melby à BYU (Brigham Young University). Il a pu intégrer un module de TA pour fournir la prétraduction. L'utilisateur peut l'accepter ou la refuser. S'il l'accepte, l'utilisateur peut demander de désambiguïser le niveau lexical et structural.

2.3.2.2.Eurolang Optimizer (1992-1996)

Le projet Eurolang Optimizer a été mentionné au 1.2.3.1.2. Le scénario était que chacun travaille sur son poste avec un module intégré à MS Office sur un PC. Quand on veut traduire un document, on l'envoie au serveur central pour prétraduction par TA et par MT. Le document prétraduit est renvoyé à l'utilisateur, après post-édition, et le document ainsi que les termes utilisés sont mis à jour sur le serveur.

Ce système a été utilisé en interne à SITE/Sonovision, puis arrêté après l'échec à l'appel d'offres de l'UE en 1995.

2.3.2.3.Prototype BEYTrans

Après avoir contribué au projet QRLex de K.Kageura (voir ci-dessus), Y. Bey a développé BEYTrans [Bey et al., 2008], un environnement de traduction collaborative en ligne comme translationwiki.net. L'idée est de construire un système Web qui permet aux traducteurs de traduire ensemble des documents. Les traducteurs travaillent sur l'ensemble des segments extraits du document, et prétraduits par un système de TA. En cours de traduction, le traducteur peut voir les suggestions de traduction des mots en question, extraites d'une base lexicale, elle aussi collaborative.

2.3.2.4.SECTra_w 2.0 dans le sous-projet EOLSS/UNL-FR de post-édition de TA

SECTra_w a été étendu pour permettre la post-édition collaborative de documents dans le cadre du projet EOLSS/UNL-FR, de la fondation UNLNL, visant à la traduction de l'anglais vers les 5 autres langues officielles de l'UNESCO (français, espagnol, russe, chinois, arabe) de 25 articles de l'EOLSS²⁸ (Encyclopedia of Life Support Systems), qui contient environ 800-880 pages, 200-220K mots. EOLSS contient au total 6. 600 articles.

Chaque article se présente comme un fichier HTML (.aspx) et un fichier « compagnon » (.unl) qui contient les segments « normalisés » pour la TA et leurs graphes UNL. Pour cette expérience,

- les 25 articles ont été prétraduits par SYSTRAN et Reverso puis post-édités en ligne,
- on a mis en service le déconvertisseur UNL-français (avec une qualité lexicale très basse car on n'avait pas assez de financement pour compléter le dictionnaire UNL-FR).

Chaque contributeur a son compte, et un « profil » défini par un administrateur du projet. Ensuite, il choisit librement des segments à post-éditer. Chaque fois qu'il prend un segment à traduire, un compteur s'active pour mesurer le temps de traduction et de post-édition sur ce segment. Une note de traduction est calculée par défaut avec le profil d'utilisateur, mais l'utilisateur peut noter sa traduction.

²⁸ <http://www.eolss.net/>, visité en juin 2009

Évolution vers les systèmes de TA hétérogènes

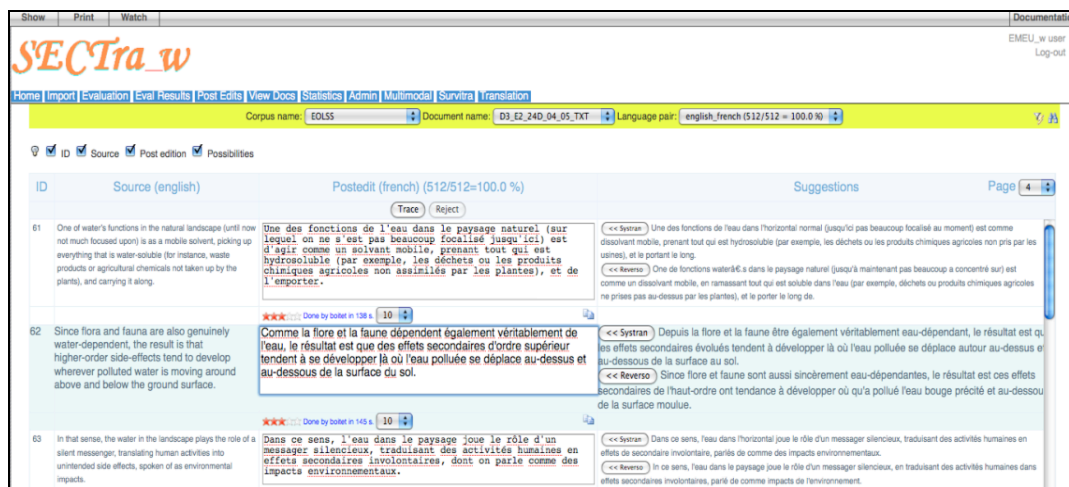


Figure 30: Interface de SECTra_w . 2.0

Pour le service de déconversion, nous sommes partis du système UNLDeco créé par Gilles Sérasset [Sérasset et Boitet, 1999] (voir partie 2.2.3.3) pour développer un moniteur Web, EMEU_w²⁹ pour à la fois fournir le résultat de traduction fourni par SECTra_w et le service de déconversion. La technique d'implémentation d'EMEU_w est inspirée de WICALÉ [Carpena, 2004], un méta-EDL générique pour systèmes de TAO. (On y reviendra dans le chapitre 4).

Une évolution actuelle est d'intégrer à SECTra_w des aides dictionnaires hétérogènes via PIVAX, qui est une base lexicale. Cette idée de contribution lexicale a été proposée et implémentée depuis longtemps [Murata et al., 2003].

Les premiers buts de la base lexicale PIVAX [Nguyen, Boitet et Sérasset, 2007] sont d'échanger des dictionnaires entre systèmes de TA avec pivot lexical (UW d'UNL) et de mettre en commun le développement si possible (U++).

Un aspect de PIVAX est de résoudre les difficultés dues à l'hétérogénéité des systèmes de TA et de TAO, un autre est la mise en service d'un environnement de développement lexical utilisable par des lexicographes et des linguistes via Internet. Pour cela, il a été construit à partir de la plate-forme générique Jibiki [Jibiki, 2009]. On reviendra à PIVAX dans la partie 3.

Cette intégration ajoute la fonction de consultation de dictionnaire par une interface simplifiée, qui est l'opération la plus importante. On la verra en détail dans le chapitre 8.

Récapitulatif

La plupart des systèmes de TH se limitent à l'appel de la TA comme une « boîte noire » pour prétraduire le texte, puis proposer aux traducteurs de réviser et/ou post-éditer. On ne peut pas utiliser les systèmes de TA de façon intégrée. Or, on pourrait les adapter ou les entraîner pour traduire mieux dans un nouveau domaine, mettre à jour des dictionnaires pour pouvoir traduire au fur et à mesure mieux qu'avant.

2.3.3.Systèmes de TA intégrant des fonctions de TH

2.3.3.1.Avant le Web 2.0

2.3.3.1.1.Ariane-78 (1978-1985), puis Ariane-G5 (1985-)

²⁹ <http://sway.imag.fr/unldeco>, visité en juin 2009, géré par Hong-Thai NGUYEN.

L'ajout de fonctions de THAM au système Ariane-78, puis Ariane-G5 a été présenté au 1.1.1. La révision est faite à la sortie de la TA sans retour au système. La collaboration était impossible à cette époque.

2.3.3.1.2. Systèmes de TA japonais (1982-)

Depuis 1982, on voit une croissance importante du nombre des systèmes de TA japonais : ATLAS-II, AS-Transac, HICATS, Duet-2, Pensée, PIVOT puis Xroad... Dans chaque système, on cherche à intégrer le système de TA dans un éditeur de traitement de texte pour en faciliter l'utilisation.

2.3.3.1.3. SPANAM & ENGSPAN (1984-)

L'application de la TA à PAHO (voir chapitre 1.1.2) est un succès illustrant l'utilisation de la TA dans la TH. L'appel de SPANAM et de ENGSPAN est intégré opérationnellement dans un flot de travaux de traduction, avec une plate-forme distribuée.

2.3.3.1.4. LOGOS avec Wang

Le système LOGOS [Scott, 1989] a été construit en 1964. La première installation réalisée vers 1966 a été utilisée par l'USAF (Armée de l'Air américaine) pour traduire d'anglais en vietnamien des milliers de pages de maintenance pendant la guerre du Vietnam. En 1978, LOGOS a obtenu un contrat pour construire un système de TA allemand-anglais avec Siemens. Cependant après 3 ans, ce système était sous-utilisé à cause de la mauvaise qualité de traduction et de la limite du financement attribué par Siemens. Après, LOGOS a signé un contrat important avec la compagnie Wang pour implémenter un meilleur système allemand-anglais (pour mainframe IBM) tournant sur des ordinateurs de bureau Wang. Ce système a été vendu à plusieurs compagnies multinationales.

Le fonctionnement de LOGOS/Wang est que, depuis un éditeur Wang en local, on a la possibilité d'entrer directement le texte à traduire, de post-éditer le résultat donné par un serveur LOGOS, et de mettre à jour le dictionnaire en ligne.

2.3.3.2. Avec le Web 2.0

2.3.3.2.1. Yakushite.net: intégration de contributions lexicales au système Pensée

L'environnement de traduction collaborative Yakushite.net (voir 1.1.3) à Oki Electric Co., Ltd (Japon) [Murata et al., 2003] offre trois fonctions principales : traduction, post-édition et gestion de dictionnaires. Les fonctions de gestion de communauté contiennent la gestion des contributeurs, la propagation des bulletins et un service de FAQ. Toutes ces fonctions sont intégrées avec le module de TA qui est fourni par le système Pensée [Shimohata et al., 1999].

Le but de Yakushite.net est de profiter de la contribution sur les dictionnaires via le Web par des communautés (on y reviendra dans la partie 3). Pour cela, on doit résoudre des difficultés comme passer à l'échelle, connecter la base de données dictionnaires contribuéés par les humains aux dictionnaires compilés du système Pensée.

Les solutions retenues pour cela sont :

- une architecture distribuée utilisant plusieurs instances du système de TA pour traduire en parallèle des segments dans un document,
- une pré-compilation des dictionnaires contenus dans la base de données vers des formats chargeables par le système de TA.

2.3.3.2.2. SYSTRANet

La société SYSTRAN a étendu la version de SYSTRAN serveur à SYSTRANet³⁰ pour le public. Un utilisateur inscrit peut déposer des documents (texte, Office, page Web, PDF) à traduire, puis appeler SYSTRAN pour une prétraduction. On peut gérer des dictionnaires utilisateur et choisir un domaine pour adapter la traduction sur SYSTRANet. L'utilisateur n'achète pas le système, mais un abonnement au service.

L'utilisateur peut faire la désambiguïsation après la phase d'analyse morphologique et faire la révision sous un environnement installé en local [Costa et Panissod, 2003].

2.3.3.2.3. Google Translator Toolkit pour traduire des documents en ligne

À partir des deux composants importants : éditeur en ligne Google Doc et le système de TA Google Translate, Google a annoncé un système de THAM intégrant la TA : Google Translator Toolkit³¹.

Après avoir importé et prétraduit par Google Translate (par une MT si elle existe, puis par TA), on peut post-éditer le document sous forme d'une page HTML. Chaque fois qu'on clique sur un segment (source), une fenêtre de post-édition de segment cible correspondant apparaît. On peut insérer des balises pour des mots particuliers.

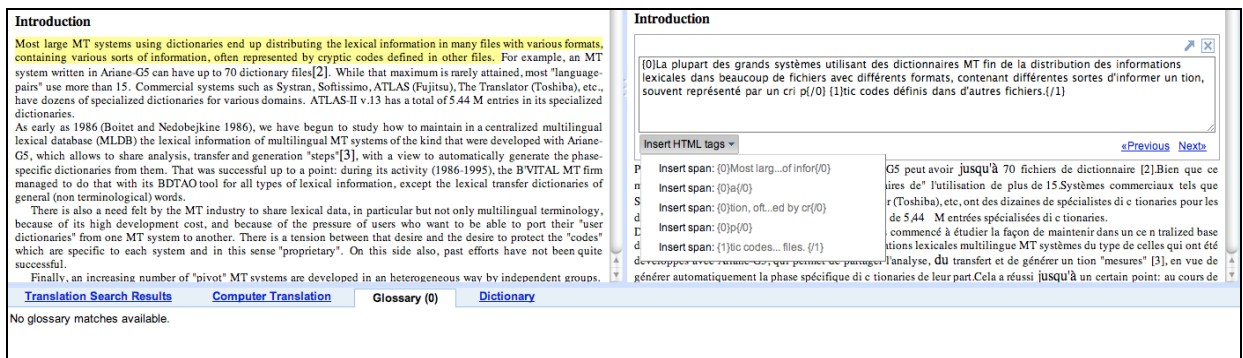


Figure 31: Interface de traduction de Google Translator Toolkit

En bas, on peut consulter le dictionnaire de base fournie par Google ou un glossaire partagé utilisé uniquement par ce document. Évidemment, on peut partager le document source et le document en cours de traduction avec d'autres utilisateurs pour traduire en mode collaboratif.

Conclusion

Dans cette partie, nous avons étudié la construction de systèmes de TA hétérogènes, qu'ils soient multiples, à composants hétérogènes, ou à participation humaine. La combinaison de systèmes existants pour former un système multiple est une voie d'avenir, surtout si l'on y intègre un bon « oracle » déterminant la « meilleure » traduction obtenue par TA, ou plus exactement la traduction la plus utile pour la compréhension ou la post-édition, selon la situation traductionnelle.

La construction de systèmes de TA multiples par combinaison de composants créés sous divers environnements et avec des méthodes et des outils différents est une approche qui se développe, à cause de la difficulté de réaliser des systèmes fortement multilingues (comme UNL) sans financement industriel permettant un développement centralisé.

Plusieurs stratégies d'intégration de travail humain et de TA ont été essayées. Créer une synergie entre TA et THAM est difficile, qu'on intègre des fonctions de post-édition à de la TA ou qu'on appelle des systèmes de TA depuis un système de THAM. Des succès relatifs

³⁰ <http://www.systranet.fr/>, visité en juin 2009

³¹ <http://translate.google.com/toolkit/>, visité en juin 2009

peuvent être obtenus si les utilisateurs (post-éditeurs) sont très proches de l'équipe de développement de la TA, sinon il faudrait pouvoir partager les ressources lexicales de TA et de THAM, ce qui n'a été jusqu'ici réalisé que par Yakushite.net.

Enfin, associer des utilisateurs humains au processus de TA lui-même est une voie à la fois possible, et nécessaire dans certains situations. Il peut s'agir de désambiguïsation interactive en source, lors de l'analyse, ou en cible, ou au niveau d'une présentation sémantique intermédiaire, dans le cas de tâches précises et de domaines restreints.

Le défi pour le futur est de permettre tous ces types d'hétérogénéité. Avant de proposer des solutions, il sera utile d'examiner plus en détail Ariane-G5, un environnement de développement et d'exploitation de TAO déjà utilisé pour plusieurs prototypes de TAO hétérogènes.

Chapitre 3. Un exemple complet : Ariane-G5 et son EDL

Introduction

L'étude de l'architecture et de la technique de réalisation d'un système de TAO hétérogène à partir de systèmes de TA homogènes devrait être appuyée par un projet réel sur un système complet qui couvrirait le mieux possible les problèmes majeurs de réingénierie logicielle et linguistique.

C'est la raison pour laquelle nous présentons dans ce chapitre le système Ariane-G5, un environnement de TA complet, et le projet Ariane-Y qui vise à construire un système hétérogène.

Des recherches en TAO³² ont été menées à Grenoble depuis 1961, sous la direction du Professeur Vauquois, d'abord dans le cadre du CETA, laboratoire propre du CNRS, puis, à partir de 1971, dans le cadre du GETA, unité de recherche associés (URA) au CNRS. À la fin de la première période, vers 1967-70, un premier système russe-français avait été développé, et testé sur plus de 300 000 mots de textes réels (articles scientifiques). Il s'agissait alors de *TAO du veilleur*, ou TA pure et dure, c'est-à-dire qu'on visait un processus purement automatique tel que le lecteur, supposé spécialiste du domaine, puisse accéder au contenu de l'original sans en connaître la langue.

À la suite d'un changement d'ordinateur (IBM 7044 à IBM 360/67), qui aurait demandé une conversion importante difficilement justifiable hors d'une perspective d'exploitation opérationnelle, ce système dut être abandonné, malgré sa qualité et son ampleur remarquables. Ce fut l'occasion de repartir sur des idées nouvelles, dans le contexte de la *TAO du réviseur*, dans laquelle on vise à produire automatiquement des traductions brutes, destinées à être révisées par un professionnel. Cette étape de révision manuelle devant naturellement se faire sur machine, ce fut aussi l'occasion d'aborder la *TAO du traducteur*, ou "traduction informatisée" (TI), dans la quelle le traducteur est post-éditeur de la TA, si elle existe.

En 1978, une nouvelle méthodologie de programmation linguistique avait été proposée (approche transfert multiniveau, programmation heuristique), et commençait à être expérimentée sur divers couples de langues, dont principalement le russe-français, grâce aux éléments logiciels d'un environnement informatique de génération de systèmes de TAO multilingues, qui fut appelé "Ariane-78" quand sa première version complète fut disponible, début 1978. Ce nom fut choisi par référence à la déesse et à son fameux fil : il s'agissait de souligner que l'informatique, si elle est essentielle, doit être mise au service de non-informaticiens et leur permettre de travailler de façon autonome, grâce à des langages spécialisés adéquats (langages symboliques de règles), et à une interface interactive transparente.

³² Les textes de cette partie sont essentiellement extraits de supports de cours préparé par Christian Boitet pour l'école d'été de Lannion organisée en 1990 par le LATL et le CNET.

Si l'auteur ne précise pas, le style de citation dans cette partie est réservé implicitement pour cette extraction. Le style *italique* est utilisé pour les termes utilisés dans Ariane-G5.

Grâce au soutien du CNRS et de la DRET, un premier prototype "préopérationnel" russe-français fut ensuite développé et expérimenté en grandeur réelle, de début 1980 à fin 1986. En parallèle, un effort de valorisation fut entrepris. En 1981-82, le GETA fut le moteur du projet ESOPE de l'ADI, dont le but était de préparer un "projet mobilisateur de la filière électronique", le "PN-TAO" (projet national de TAO). De 1983 à 1987, le GETA y participa très activement, en augmentant les capacités et la fiabilité d'Ariane-78.4, et en étant au cœur de la spécification linguistique du système français-anglais pour l'aéronautique construit en coopération avec les partenaires industriels.

Au GETALP, il y a le projet Ariane-Y en cours. Il s'agit de la génération suivante du système Ariane-G5.

Pour étudier la méthodologie de construction d'un système de TA et la réingénierie de tels systèmes vers un système de TAO hétérogène, il nous faut participer à un projet réel en cours de réalisation d'un système complet et complexe tel que le projet Ariane-Y.

Dans la première partie, on présente brièvement les principes généraux du système Ariane-G5. Ensuite, les parties 2 et 3 présentent ses composants logiciels et linguistiques. Le projet Ariane-Y et ses principes de réalisation sont présentés dans la partie 4.

3.1.Principes généraux

Ariane-G5 est un générateur (G) de systèmes de TAO reposant sur cinq (5) langages spécialisés pour la programmation linguistique (LSPL). Chacun de ces langages est compilé, et les tables internes produites sont données en paramètres au "moteur" du langage.

Bien qu'il soit particulièrement adapté à l'approche transfert, il ne l'impose pas. À part les limites d'implémentation, la seule contrainte forte est que les structures de représentation des unités de traduction doivent être des arbres décorés.

La sémantique intrinsèque (terme emprunté à J. P. Desclés) est représentée dans ces langages, donc de façon linguistique. Si l'on veut écrire un système spécialisé à un sous-langage réduit et à un microdomaine, rien n'interdit de suivre la même voie que METEO [Chandioux et Guéraud, 1981] et d'écrire des grammaires et des dictionnaires "sémantiques". Pour construire un système muni d'une sémantique extrinsèque ("ontologie" de l'univers de référence), il faudrait coupler Ariane-G5 à un "système expert correcteur", comme cela a été suggéré et prototypé dans [Gerber, 1984] (voir 2.2.1).

Par rapport à la quasi-totalité des systèmes existants, Ariane-G5 présente l'avantage que l'unité de traduction n'est pas réduite à la phrase, mais peut comporter plusieurs paragraphes (100 à 200 occurrences en général, soit près d'une page standard).

3.1.1.Environment matériel et logiciel

3.1.1.1.Système d'exploitation et plates-formes

Ariane-G5 a tourné sous VMSP/CMS, sur gros ordinateurs (3090, 303X), sur mini (43XX, 937X, 9221, H30), et sur gros micro (PC-AT/370, PS2-80/7437).

Depuis 2001, Ariane-G5 tourne sur un IBM H30 sous z/VM (voir Table 2).

zVM (successeur de CP puis VMSP) est un hyperviseur qui simule un ensemble de machines virtuelles. Chaque machine virtuelle tourne avec un système d'exploitation propre. Par exemple, on peut faire tourner en même temps des

machines virtuelles sous VMS/TSO, sous CMS, sous AIX, et sous Linux/390. CMS est un système d'exploitation interactif mono-utilisateur puissant, qui supporte un grand nombre de langages et d'outils de programmation.

On pourrait effectivement faire tourner Ariane-G5 sous Linux/390, mais il faudrait modifier les routines « système » et les recompiler, et revoir les programmes en PL/I.

3.1.1.2. Instances installées

Le système Ariane a été implanté dans de nombreux sites pour des raisons diverses (coopérations scientifiques, travaux et démonstration du GETA).

Table 2: Installations d'Ariane

Organisation/Université	Plate-forme	Année
Sparkasse de Saarbrücken	370/158	1981
Réseau IBM-bureau	370/158	1979-1980
Réseau CISI	Vélizy, 3031	1979-1980
Université de Louvain la Neuve	370/168	1979
USM, Penang	370/148	1979-1987
Université de Leuven	4331	1979-1980
Caisse régionale du Crédit Agricole à Grenoble	3032	1979-1980
Université de Campinas BR	370/148	1979-1981
Université Chulalongkorn, Bangkok	370/138	1980-1989
Société IEE, Paris	NASCO AS 3/5	1980
Association Champollion	4331, 4361, 4381, 9221	1981-2000
Société SG2, Garges-les-Gonesse	370/158	1982-1987
NHK, KDD (Japon)	-	1983-1988
CAS, Académie des sciences chinoise	-	1983-1990
CIST, Chine	-	1983-1990
BYU, Provo, Utah, USA	-	1984-1985
Institut Académique de Nankin, Chine	-	1987-1990
IERA (Rabat)	370/158	1993
Université St Jérôme (Marseille)	-	1996-2000
IBM (Montpellier)	-	1999
GETA-CLIPS	H30 (labo), 1Go 60Mip + VM/Linux 390	2001-2006
GETALP, LIG		2007-

Depuis 1990, on ne cherche plus à installer le système partout. On est passé à l'idée d'utiliser le système comme un serveur central utilisé à distance non seulement pour l'exécution (réseau LIDIA, voir 3.2.2) mais aussi pour la préparation (CASH et WICALE voir 4.1). Les installations à Marseille (Université St-Jérôme) et à Montpellier (IBM) ont été faites pour bénéficier de processeurs très rapides en vue des expériences et démonstrations de traduction de parole des projets CSTAR-II (1995-1999) puis Nespole! (1999-2003).

3.1.1.3. Environnement actuel

La version actuelle d'Ariane-G5 représente environ 400 000 lignes de programmes sources, plus 30 000 lignes de messages par langue de dialogue. La partie exécutable réside sur un « minidisque » (disque virtuel) d'une machine virtuelle particulière, et occupe environ 10 Mo.

Pour rendre Ariane-G5 accessible à distance, P. Guillaume a développé le réseau LIDIA (voir 3.2.2) pour supporter l'échange des commandes et des données par les protocoles : SMTP, HTTP, et socket.

Pour travailler sous Ariane-G5, un linguiste utilise en général 3 minidisques: A (minidisque utilisateur), B (minidisque d'Ariane avec certains programmes modifiés), C (la version stable du système).

Pour qu'une machine virtuelle puisse être une « machine utilisateur » (d'Ariane), il suffit d'effectuer une connexion logique (en « B/A » ou « C/A ») du minidisque d'Ariane à son minidisque (« A »). Ariane gère alors, sur le minidisque utilisateur, deux bases de données spécialisées. L'une contient les linguiciels (grammaires, dictionnaires, *etc.*), et l'autre les textes (au maximum 1 000 langues sources et cibles, 1 000 « corpus », et 10 000 textes par corpus).

La formation informatique minimale nécessaire pour utiliser Ariane-G5 consiste à apprendre les commandes élémentaires de début et fin de session (login/logout), l'éditeur d'écran XEDIT, et pour les développeurs de systèmes de TAO, l'organisation du moniteur interactif et les langages spécialisés.

3.1.2. Organisation logique

3.1.2.1. Étapes, phases et articulations du processus de traduction

3.1.2.1.1. Ariane-78

Le processus de traduction se compose de trois étapes (Analyse, Transfert, Génération) et chaque étape est elle-même composée de phases, une phase étant écrite dans un langage spécialisé pour la programmation linguistique (LSPL) particulier. Dans Ariane-78, il y avait 4 LSPL : ATEF, ROBRA, TRANSF et SYGMOR.

Le format intermédiaire entre deux phases successives est une liste d'arbres décorés (voir 3.3.2.3.1) correspondant à des unités de traduction successive.

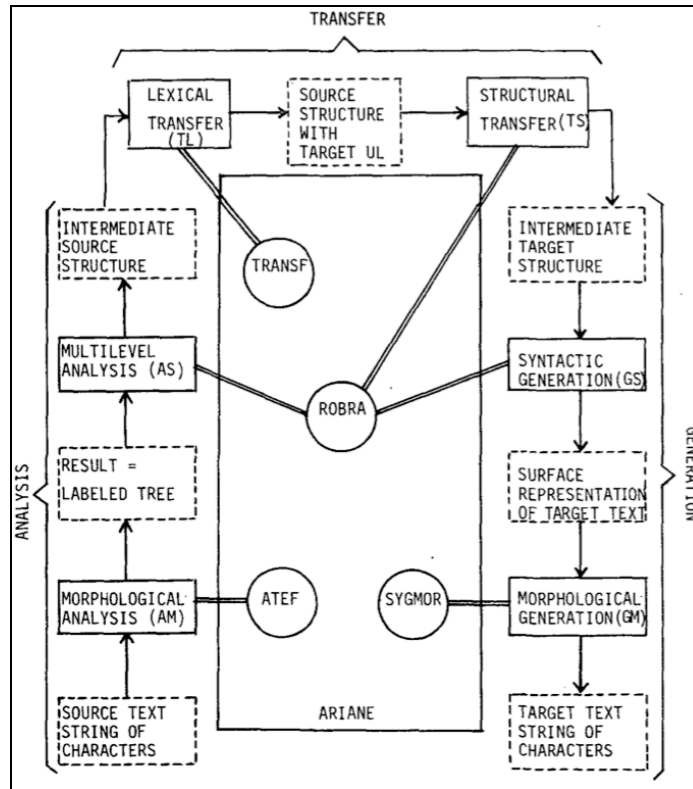


Figure 32: Étapes/Phases dans Ariane-78

3.1.2.1.2. Ariane-G5

À la demande des développeurs (académiques et industriels), on a introduit dans Ariane-G5 (entre 1985 et 1988) des phases facultatives (AY, AX, TX, TS, GX, GY) en plus des phases obligatoires (AM, AS, TL, TS, GS, GM). L'utilisateur peut définir la chaîne d'exécution ou la chaîne de production en choisissant les phases nécessaires.

Les composants linguiciels sont définis et développés selon la syntaxe d'un des 5 LSPL. Chaque phase utilise un LSPL : ATEF (AM), EXPANS (AX, AY, TL, TX, TY, GX, GY), ROBRA (AS, GS), SYGMOR (GM), et TRACOMPL pour écrire les « articulations » possibles : AMAX, AMAS, AYAS, AXAS, ASTL, et puis TLTS, TXTS, TSTY, TYGX, TSGX, TSGS, TYGS, et enfin GXGS, GSGM, GSGY, GYGM.

Une opération linguistique donnée n'est pas nécessairement centralisée uniquement dans la phase qui porte son nom (comme AM pour l'analyse morphologique), mais peut être distribuée en plusieurs phases. Par exemple, l'analyse morphologique ne se limite pas nécessairement à la phase d'AM, mais dans les phases AX, AY, et éventuellement dans une fraction d'AS (par exemple, pour connaître la présence effective de tournures non connexes). En général, le transfert lexical est réparti (au moins) entre TL et TS. La génération morphologique d'une langue peut être répartie sur la fin de la phase GS et sur la phase GM.

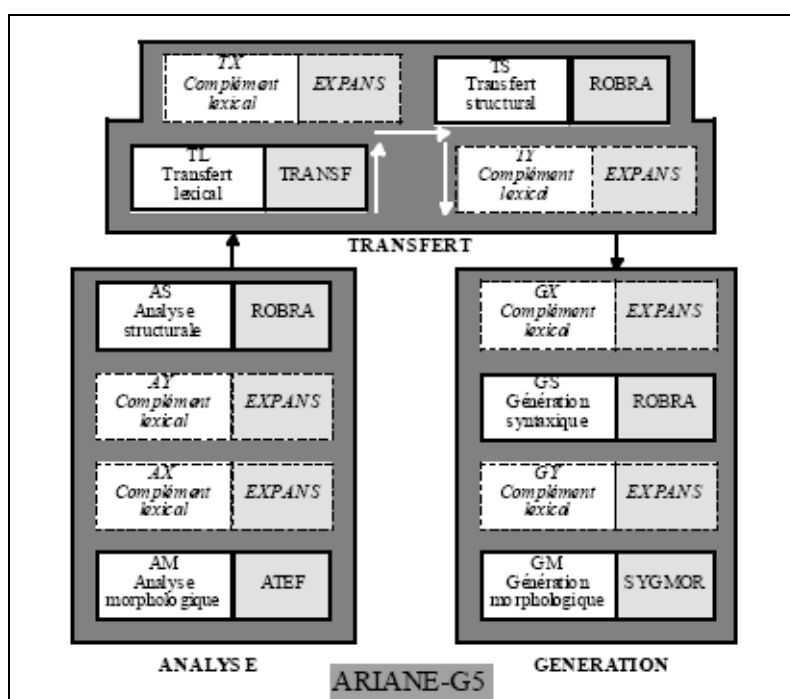


Figure 33: Étapes/Phases dans Ariane-G5

En analyse, on trouve :		
•AM	Analyse morphologique	obligatoire
•AX	Analyse Expansive	facultative
•AY	Analyse Expansive	facultative
•AS	Analyse Structurale	obligatoire
En transfert, on trouve		
•TL	Transfert Lexical	obligatoire
•TX	Transfert Expansif	facultative
•TS	Transfert Structural	obligatoire
•TY	Transfert Expansif	facultative
En génération, on trouve		
•GX	Génération Expansive	facultative
•GS	Génération Syntaxique	obligatoire
•GY	Génération Expansive	facultative
•GM	Génération Morphologique	obligatoire

3.1.2.2. Types de données essentiels et terminologie de base

À l'entrée et à la sortie du processus de traduction, l'unité de traduction est une simple chaîne de caractères. Les 256 caractères EBCDIC sont considérés comme atomiques (non structurés), et tous sont admis pour former les chaînes des langages spécialisés. Le blanc (X'40') est utilisé comme séparateur d'occurrences. Une unité de traduction est donc aussi une suite d'occurrences.

De la sortie d'AM à l'entrée de GM, l'unité de traduction est représentée par un arbre décoré. Chaque phase comporte une partie où le linguiste déclare le *type de décoration*, ou *jeu de variables* dans la terminologie d'Ariane-G5.

Une *variable élémentaire* (variable pour abrégé) est définie par un nom et un type (anonyme). Ce type est défini par un *genre* (exclusif, non-exclusif, arithmétique, lexical) et une liste de valeurs. Chaque variable a une valeur nulle dénotée par son nom suivi de "0".

- Une variable exclusive V définie par $V == (V1, V2, V3)$ prend ses valeurs dans $\{V0, V1, V2, V3\}$. On peut faire une analogie avec les types scalaires de Pascal, à ceci près que les identificateurs des valeurs sont ici locaux aux types.
- Une variable non-exclusive V définie par $V == (V1, V2, V3)$ prend ses valeurs dans les parties de $\{V1, V2, V3\}$. V0 dénote alors l'ensemble vide. On peut faire l'analogie avec les types ensembles de Pascal, avec la remarque précédente.
- Une variable arithmétique V définie par $V == (n)$, où n est un entier naturel non nul, prend ses valeurs dans :

$$\left[-2^{\lfloor \log_2(n) \rfloor}, 2^{\lfloor \log_2(n) \rfloor - 1} \right]$$

- Il y a toujours une (et une seule) variable "lexicale", de nom UL, pour unité lexicale, qui est prédéclarée, et prend ses valeurs dans l'ensemble formé :
 - des valeurs prédéfinies " (UL0), 'ULTXT', 'ULFRA', 'ULSOL', 'ULOC', 'ULMCP' ;
 - des valeurs introduites dans les composants linguistiques (essentiellement dans les dictionnaires) ;
 - des valeurs construites dynamiquement (par exemple pour les mots inconnus).

Une *décoration*, ou *masque de variables* en jargon Ariane-G5, est une combinaison de valeurs de toutes les variables du jeu de variables considéré, et est donc tout à fait semblable à une liste de propriétés en LISP.

On peut regrouper les variables de façon hiérarchique, le sommet de la hiérarchie étant prédéclaré jusqu'à un niveau dépendant du langage spécialisé. VAR dénote toujours le jeu de variables diminué de l'UL.

En ATEF, on distingue deux sous-ensembles, VARM et VARS, déclarés séparément, pour les variables dites "morphologiques" et "syntaxiques" (bien que, comme pour les phases, les linguistes ne respectent en général pas cette division, et ajoutent un bon nombre de variables de nature sémantique). VARM et VARS sont divisés en VAREM et VARNM, VARES et VARNs (exclusives et non-exclusives), car il n'y a pas de variables arithmétiques en ATEF.

En ROBRA et EXPANS, où les trois genres de variables sont possibles, le sommet de la hiérarchie est VAR (VAREp , VARNp , VARAp), où p est un caractère (redéfinissable dans DV) caractéristique de la phase. Par défaut, p vaut S, R, C, D, G pour AS, TL, TS, GS, GM, et X pour les phases EXPANS. Par exemple, en AS, pour raffiner VARE en variables syntaxiques et sémantiques, elles-mêmes divisées en propriétés et relations, on pourrait écrire :

```
-EXC-** (mot-clé pour "exclusif").
  VSYNTE == (PSYNTE(CAT(N,V,A,R,S...),K(PHVB,PHINF,GV,GN,GA)),
            RSYNTE(FS(SUJ,OBJ1,OBJ2,EPIT,CIRC...))).
  VSEME == (PSEME(PREDIC(ETAT,ACTION,PROC),MATIERE(DISC,CONT)),
            RSEME(RL(ARG0,ARG1,ARG2,ARG01,ARG02,TRL10...))).
-NEX-** non-exclusives.
```

Un *format* est un masque de variables constant auquel on a donné un nom, pour l'utiliser comme abréviation dans les dictionnaires et les grammaires. Un arbre décoré est un arbre orienté et ordonné dont chaque nœud porte une décoration.

3.1.2.3. Composants et variantes d'une phase

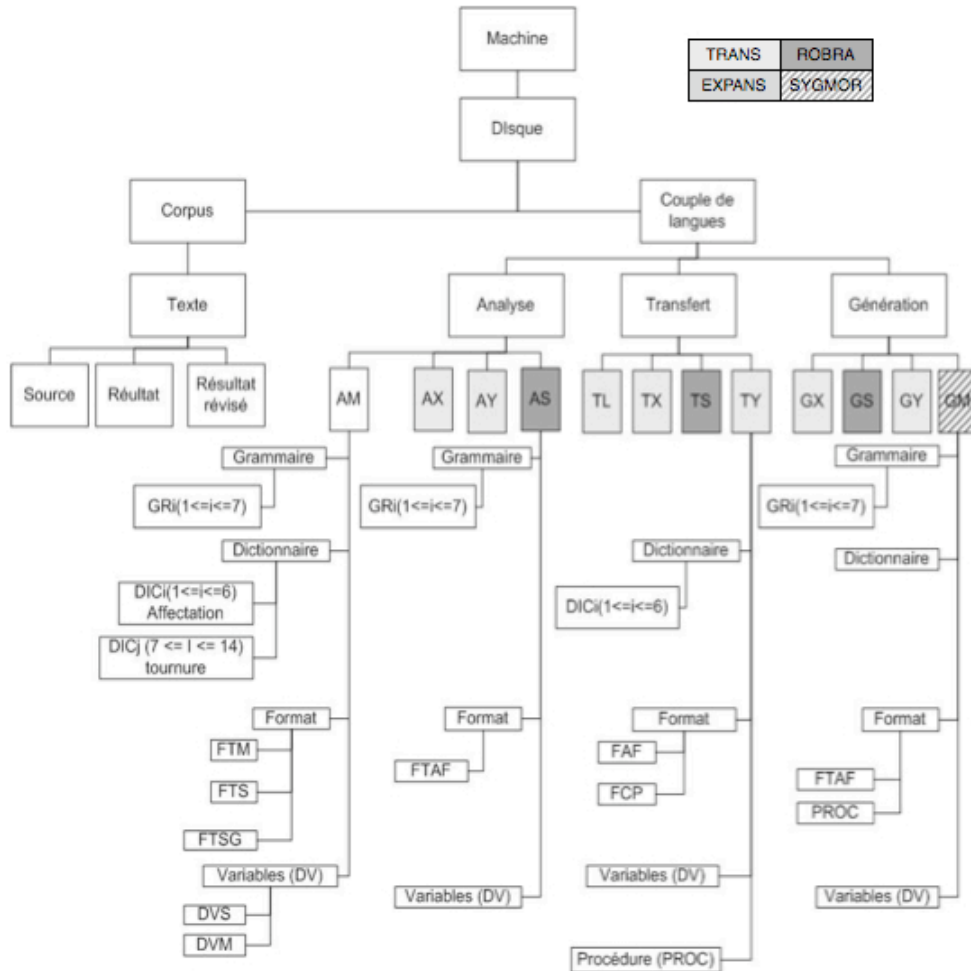


Figure 34: Composition des logiciels d'Ariane-G5

DIC	Dictionnaire
GR	Grammaire
FTM, FTS	Formats « morphologiques » et « syntaxiques »
FTSG	Formats « syntaxiques généraux »
FAF	Formats « d'affectation »
FCP	Formats « de conditions propres »
PROC	Procédures de conditions, de conditions et d'affectation
DV	Déclaration des variables
DVM, DVS	Déclaration des variables « morphologiques » et « syntaxiques »

Comme dans la plupart des systèmes de TALN, les langages spécialisés sont organisés en *composants* physiquement distincts, pour des raisons de modularité et de taille. Les composants d'une phase forment un graphe acyclique de dépendance (pour la compilation).

- Une phase ATEF comporte deux composants de déclaration de variables (DVM, DVS), des formats morphologiques, syntaxiques et "généraux"

(FTM, FTS, FTSG, ce dernier composant étant optionnel), 1 à 7 grammaires GR_i (1 ≤ i ≤ 7), 1 à 6 dictionnaires de "morphes" DIC_i (1 ≤ i ≤ 6), dont au moins un de "bases", et de 0 à 7 dictionnaires de tournures figées connexes DIC_i (7 ≤ i ≤ 14). FTM dépend de DVM, FTS de DVM et DVS, FTSG de FTS, les dictionnaires des formats, et les grammaires des dictionnaires.

- Une phase EXPANS comporte un composant de déclarations de variables (DV), un de procédures de condition et d'affectation sur les décorations (PROC), un de formats d'affectation (FAF), optionnellement un de "formats de conditions propres" (FCP), et de 1 à 7 dictionnaires (DIC_i). Il existe aussi un dictionnaire DIC₀, optionnel, utilisé dans le cas où une UL est « inconnue » dans les dictionnaires « normaux ». Il contient des articles normaux et un article spécial permettant de construire une UL cible à partir de la valeur de chaîné d'une UL source inconnue. PROC, FAF et FCP dépendent de DV, et les DIC_i des précédents.
- Une phase ROBRA comporte DV, FAF, et de 1 à 7 grammaires (GR₁ à GR₇), avec les dépendances naturelles.
- Une phase SYGMOR comporte DV, FAF, PCP (procédures de conditions propres), GR_i (1 ≤ i ≤ 7), et DIC_i (1 ≤ i ≤ 13, un dictionnaire au moins étant accédé par l'UL), avec les mêmes dépendances qu'ATEF, sauf que les grammaires ne dépendent pas des dictionnaires.
- Une articulation TRACOMPL ne comporte qu'un composant, DV.

L'interface Ariane-G5 assure à tout instant la cohérence des tables internes en fonction de ces dépendances et des modifications effectuées par l'utilisateur.

Chaque phase peut donner lieu à des *variantes*, qu'on peut définir en fonction des types de textes à traduire..

- En ATEF et en SYGMOR, on choisit une des grammaires.
- En EXPANS, on choisit un sous-ensemble ordonné des dictionnaires, et le mode déterministe ou non du moteur.
- En ROBRA, on définit une liste d'au plus 14 grammaires à exécuter séquentiellement, la même pouvant apparaître plus d'une fois.

En combinant ces choix et le choix d'un chemin dans le graphe (de AM à GM pour une traduction), on obtient ainsi des *chaînes d'exécution* (pour la mise au point) et des *chaînes de production* (pour l'exploitation) qui sont elles aussi mémorisées et gérées par Ariane-G5.

3.2. Composants logiciels

3.2.1. Moniteur interactif natif

Le moniteur d'Ariane [Quézel-Ambrunaz, 1989] est un système conversationnel du type question-réponse, qui guide l'utilisateur pour choisir parmi les différents traitements proposés. Une fois sous Ariane-G5, on peut :

- travailler sur les composants linguistiques (phases, articulations...), dans les sous-environnements PRAM, PRGM, PRAMAX (création, modification, compilation, listage, ...).
- travailler sur les corpus (PRCORP), et les textes (PRTEXT) avec plusieurs possibilités.

- travailler sur les chaînes d'exécution (PRCHEX) et de production (PRCHPROD).
- Exécuter (EX<ph>) tout ou une partie (EXECUT, EXPROD) d'un processus de traduction pour le mettre au point (chaque phrase a des paramètres pour tracer à plusieurs niveaux et récupérer les formats de sortie) à l'aide d'une chaîne d'exécution disponible.
- produire des traductions brutes en mode d'exploitation, en utilisant un contrôle d'exécution (PRTREX).
- effectuer la révision de traductions brutes (voir 1.1.1) avec l'éditeur multifenêtre XEDIT. Il était également prévu de pouvoir faire réviser les arbres de chaque phase sous l'éditeur TTEDIT [Durand, 1988]³³.
- effectuer des traitements sur plusieurs phases (par exemple, compiler de AM à TS).
- obtenir des informations sur les données gérées par Ariane (liste des langues sources et cibles, état des compilations, ...)
- changer les valeurs de paramètres globaux : langues d'interface, langue source et cible courante, corpus courant, ...)

À tout moment, on peut accéder à l'aide en ligne disponible. Ce moniteur représente environ 50 000 lignes d'EXEC2 et 100 000 lignes d'ASM370.

É. Blanc [Blanc, 1999a] a développé une interface plus conviviale appelée CASH (Commande d'Ariane Sous Hypertexte) fonctionnant sous Revolution³⁴ et basée sur l'utilisation de connexions par sockets.

Dans le cadre de son mémoire d'ingénieur CNAM, V. Carpena [Carpena, 2004] a développé WICALE (Web Interface for Communication with All Lingware Environments) version 1.0, qui permet un pilotage générique des commandes et données pour plusieurs systèmes de TALN (Ariane-G5, PILAF, ...). Ce système est implémenté en Java et utilise une connexion par sockets. Je l'ai amélioré et étendu (WICALE 2.0) [Nguyen, 2005] pour ajouter une fonction d'édition et de navigation des linguiciels en local.

3.2.2. Le réseau LIDIA

Maurice Quézel-Ambrunaz et Pierre Guillaume [Guillaume, 2003] ont implémenté le réseau LIDIA comme une des composantes du projet LIDIA du GETA (voir 2.2.2.3).

La « maquette LIDIA » est système de TA fondée sur le dialogue avec le rédacteur réalisé en deux parties : l'une tourne sur Macintosh pour le dialogue de désambiguïsation, l'autre tourne sur un serveur IBM, pour toutes les phases d'Ariane-G5.

La communication entre les machines virtuelles a été faite par le mécanisme appelé « spool » ou canal. Chaque machine range le résultat ou/et commande dans un spool pour qu'une autre machine puissent les prendre et continuer dans la suite des traitements.

L'organisation du réseau LIDIA unifie au niveau logique et physique les machines connectées. On étudie ces deux niveaux dans les parties suivantes³⁵.

³³ Cette possibilité n'a pas été intégrée dans la version actuelle (départ de M. Quézel-Ambrunaz pour cause de maladie)

³⁴ <http://www.runrev.com>

³⁵ Le texte de cette partie est essentiellement extrait de la documentation du réseau LIDIA rédigée par Pierre Guillaume en 2003 avant son départ.

3.2.2.1. Composants logiques

Côté hôte IBM, on dispose de l'organisation en machines virtuelles offerte par VM/SP (Arianenet), de leur fonctionnement en temps partagé, et de plusieurs niveaux d'interconnexion possibles des machines serveurs d'Ariane-G5, ainsi que des machines d'interface permettant l'accès au réseau de machines virtuelles à partir de divers environnements réseau physiques, en particulier TCP/IP. C'est ainsi que sont mis en service des accès via le courrier électronique (SMTP), le Web (HTTP), et Telnet (implémenté par des sockets sur la machine virtuelle interface).

La réalisation du réseau LIDIA devant se faire rapidement et sans avoir à développer un applicatif trop volumineux, on a choisi comme dispositif d'interconnexion le « Spooling System » de VM et la mise en communication des lecteurs perforateurs (READER PUNCHER) des machines virtuelles d'un même site VM/SP. Il s'agit donc de communications asynchrones et les problèmes de conflit d'accès sont laissés à VM/SP.

La communication entre machines virtuelles se présente donc comme des échanges de fichiers (fichiers SPOOL) et la gestion de files d'attente (sur des lecteurs virtuels).

Il s'est également avéré indispensable d'envisager une gestion de messages permettant une synchronisation rapide de l'état du réseau (machines serveurs Ariane-G5 invitées à stopper en fin de leur traitement courant par exemple).

Le réseau LIDIA intègre la (ou les) machines virtuelles gérant le « Remote Spooling » de façon à pouvoir éventuellement prolonger ce réseau vers des sites extérieurs. On retiendra donc que, du côté de l'hôte IBM, le réseau LIDIA se présente comme un ensemble de machines virtuelles spécialisées et interconnectées par l'intermédiaire du gestionnaire du "Spool" de VM.

3.2.2.2. Organisation des machines virtuelles sur l'hôte IBM sous VM puis zVM

Ces machines sont de trois types différents :

- Le premier type développé au tout début, en 1993, correspond à une machine d'interconnexion physique IBM/Macintosh. Elle est activée lors des échanges entre le poste Mac et l'hôte IBM. Elle est liée à un accès précis, donc à un Macintosh équipé d'une carte adaptateur. Elle est activée à la demande à partir du Mac. Elle gère les échanges dans le sens hôte IBM à Mac, avec constitution d'une file d'attente vers le Mac, et dans le sens Mac à hôte IBM, avec échange vers une machine virtuelle administrateur réseau LIDIA (troisième type).
- Le second type correspond aux machines virtuelles « serveurs Ariane-G5 ». On peut les considérer comme des machines "batch" travaillant en tâche de fond sur l'hôte IBM. Chacune de ces machines fait tourner un applicatif Ariane-G5 bien déterminé (une ou plusieurs phases) des données linguistiques adaptées (qu'elles peuvent partager en lecture pour obtenir plusieurs réalisations en parallèle d'un même applicatif). Ces machines sont activées de façon continue avec le réseau LIDIA. Elles consultent l'état de leur lecteur d'entrée, prennent en charge le traitement précisé avec les données associées, transmettent les résultats, via leur perforateur virtuel, vers une machine virtuelle administrateur du réseau LIDIA (troisième type). Si leur lecteur d'entrée est vide, elles se mettent en attente d'un événement sur ce lecteur. Elles peuvent aussi intercepter des

messages leur ordonnant un arrêt différé (fin du traitement courant) ou immédiat.

- Le troisième type correspond aux machines virtuelles "administrateur de réseau LIDIA". Ces machines centralisent tous les échanges dans le réseau et ordonnancent les traitements demandés. Elles servent toujours d'intermédiaire entre les machines des deux premiers types qui ne peuvent communiquer directement entre elles.

Les deux premiers types de machine sont propres au fonctionnement du réseau dont le but est d'offrir les services des machines Ariane-G5. Ariane-G5 apparaît essentiellement comme une séquence de phases de traitements linguistiques précis (analyseur morphologique, analyseur syntaxo-sémantique multi-niveau, transfert vers plusieurs langues cibles, *etc.*).

Chaque machine du troisième type implémente un traitement élémentaire LIDIA. Les caractéristiques de ces traitements sont connues des machines du deuxième type. Des traitements plus complexes sont demandés au niveau des machines du premier type (éventuellement à partir d'autres machines virtuelles utilisateurs d'Ariane).

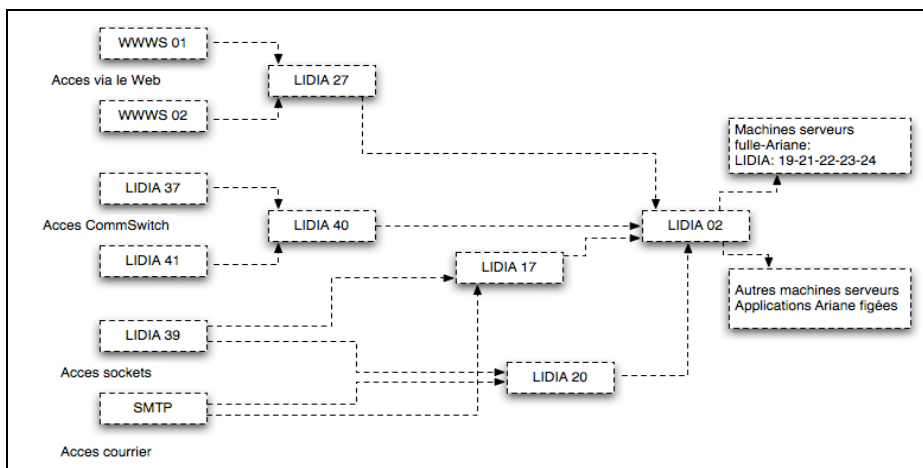


Figure 35: Organisation de principe des machines virtuelles constituant le réseau LIDIA

Le rôle des machines du deuxième type est d'assurer une prise en charge correcte de ces traitements. Pour cela, elles assurent une succession de communications avec les machines du premier et du troisième type. L'état d'avancement de ces traitements fait partie du descriptif des données qui circulent ainsi d'une machine à l'autre. En principe, les machines du deuxième type ne possèdent en propre que des informations de nature statique (constitution du réseau, nature des traitements élémentaires). Les informations dynamiques (état avec données) voyagent avec les fichiers du spool.

Dans la mesure où la tâche de ces machines consiste essentiellement en des échanges de fichiers spool, on peut se contenter d'un exemplaire unique d'une telle machine pour un réseau. L'organisation du réseau, pour des raisons de généralité, permet d'en prévoir plusieurs. Ces machines comportent des descriptifs des fonctionnalités des réseaux, et ces descriptifs sont supposés non seulement non contradictoires, mais identiques (machines symétriques banalisées). Ces machines n'ont donc pas besoin de communiquer entre elles. On peut s'adresser à une machine administrateur particulière qui peut déterminer un sous-réseau (par types de postes par exemple). Reste la (ou les) machines

"RSCS" qui sont considérées par le réseau LIDIA comme des serveurs analogues aux machines du second type (serveurs Ariane-G5), ces serveurs établissant des communications vers des sites VM (ou autres) distants (où l'on pourrait éventuellement trouver d'autres réseaux LIDIA). La notion de site (NODEID) est donc rendue explicite dans un réseau LIDIA.

Le réseau LIDIA gère les commandes arrivantes comme un flot. On peut visualiser l'état des commandes reçues sur l'écran de la machine virtuelle spéciale LIDIA00 :

```
14:13:32 * MSG FROM LIDIA17 : --> From: LIDIA39 at CLMP3000 <--
14:13:33 * MSG FROM LIDIA17 : --> 107309 LIDIA17 entr{ en r{seau et O.K.
14:13:33 * MSG FROM LIDIA34 : --> LIDIA34 O.K.
14:13:33 * MSG FROM LIDIA34 : --> 107305 LIDIA17 in.
14:13:34 * MSG FROM LIDIA34 : --> LIDIA34 : -- Ariane-G5 en {tat d'erreur --
14:13:34 * MSG FROM LIDIA34 : <-- LIDIA34 Code = 1008 .
```

Ici, on voit qu'une commande a été envoyée depuis la machine LIDIA17 (machine de développement du déconvertisseur UNL-FR). Cette commande a été communiquée à la machine serveur Ariane-G5 LIDIA 34, qui a renvoyé une erreur.

3.2.2.3. Implémentation

On a cherché à éviter un investissement trop massif pour le fonctionnement de ce réseau. Les machines serveurs Ariane-G5 (du deuxième type) ont été complétées par une couche simple à réaliser (mode « batch » du moniteur Ariane). Les machines du premier et du troisième type ont été développées essentiellement en REXX (environ 2 000 lignes).

3.2.3. LSPL

3.2.3.1. Nature, raisons de leur variété

Les LSPL (Langages Spécialisés pour la Programmation Linguistique) sont conçus pour implémenter les outils linguistiques. Au CETA, l'idée de construire un système de TALN basé sur plusieurs LSPL est venue des travaux de B. Vauquois, G. Veillon, J. Veyrunes et leurs collègues des années 1962-1967. Cette idée s'est également imposée dans [Hutchins et Somers, 1986] ou encore [Arnold et al., 1994].

La clarification de la nature et des raisons de la variété des LSPL dans un système de TA comme Ariane-G5 est utile pour comprendre la nécessité ou au moins la grande utilité de l'évolution vers un système hétérogène universel.

3.2.3.1.1. Nature

Un LSPL est un langage spécialisé conçu pour une certaine classe d'applications linguistiques. Dans Ariane-G5, on a ATEF pour l'analyse morphologique, SYGMOR pour la génération morphologique, ROBRA pour la transformation d'arbres décorés, EXPANS/TRANSF pour l'expansion lexicale monolingue et le transfert lexical bilingue, et TRACOMPL pour le « changement de coordonnées » (ou « articulation »), qui consiste à transformer un arbre décoré sur un jeu de variables vers le même arbre (du point de vue géométrique) sur un autre jeu de variables. Il y a des LSPL auxiliaires comme ATLAS pour l'aide à l'indexage des dictionnaires dans un LSPL, et LT pour l'écriture de transcripteurs.

Chaque LSPL offre des structures de contrôle et de données adaptées à son objet et à une certaine classe de méthodes algorithmiques.

3.2.3.1.2. Raisons

Spécification : Pour réaliser une tâche spécifique, on peut utiliser des langages de programmation généraux, mais ils sont, en général, trop puissants et trop peu contraints pour être adaptés au TALN [Lafourcade, 1994]. Un peu mieux, on peut utiliser un LSPL général

comme les systèmes-Q au lieu de 5 LSPL dans Ariane-G5. Cependant, l'expérience prouve que certaines tâches sont impossibles à programmer de façon efficace (en temps de calcul, et même de développement) avec des LSPL trop généraux.

Donc, un langage spécialisé associé à une tâche spécifique fournit un niveau d'abstraction plus élevé (et plus adéquat) que les langages algorithmiques et les LSPL généraux, tout en permettant plus d'efficacité.

Séparation : un LSPL est en fait un métalangage pour écrire des linguiciels (grammaires, dictionnaires, ou description de chaîne de traduction...) implémenté de façon indépendante des applications qu'il permet d'écrire.

Adéquation : Dans un LSPL, les linguistes travaillent directement avec des concepts familiers comme les variables grammaticales, les classes, les dictionnaires, les grammaires, associés à la tâche (générique) pour laquelle est conçue ce LSPL (voir 3.2.3.2 pour les exemples de chaque LSPL dans Ariane-G5).

3.2.3.2. Les LSPL d'Ariane-G5

On distingue les 5 LSPL d'Ariane-G5 par la classe des objets traités : chaîne en morphologie (analyse sous ATEF, génération sous SYGMOR), et arbre décoré (articulation sous TRACOMPL, transformation sous ROBRA, expansion lexicale monolingue sous EXPANS et transfert lexical bilingue sous TRANS).

3.2.3.2.1. AM & GM

3.2.3.2.1.1. ATEF, un langage pour l'analyse morphologique

ATEF a été conçu en 1971 par J. Chauché [Chauché, 1975], qui a écrit le moteur, tandis que P. Guillaume et M. Quézel-Ambrunaz écrivaient les compilateurs des différents composants. Depuis, un certain nombre d'extensions ont été apportées, mais le modèle algorithmique sous-jacent n'a pas varié. Il s'agit en effet d'un outil très satisfaisant.

Le système traite successivement chaque occurrence du texte, en examinant a priori toutes les analyses possibles (mode non-déterministe total avec rétrogression). La forme courante est notée C. Un résultat d'analyse est une décoration ou une suite de décorations (cas des mots composés). Chaque étape d'une analyse particulière consiste à choisir un des dictionnaires ouverts, à y trouver un article dont la clé, ou morphe, soit préfixe (ou suffixe, en mode droite-gauche) de ce qui reste à analyser (noté A), qui est diminué d'autant, et à appliquer l'une des règles associées au format morphologique de l'article en question.

Par exemple³⁶, pour l'analyse morphologique de l'allemand :

OCCURRENCE : BAUERLAUBNISSE
DIC1 : BAUER (das Bauer, le cage)
DIC2 : BAU (base verbale ; appel à R1) ;
BAUER (der Bauer, le paysan ; appel à R1 et R2) ;
ERLAUB (base verbale ; appel à R1) ;
ERLAUBNIS (base de nom ; appel à R3) ;
LAUB (base de nom ; appel à R1 et R2) ;
NISSE (base de nom ; appel à R1 et R2).

³⁶ Exemple extrait du DSE-1 d'Ariane-G5, fait par J-P. Guilbaud

- DIC3 : SE (désinence de nom).
 GRAMMAIRE : - RDICT ouvre au départ les dictionnaires 1 et 2 ;
 - R1 traite les bases lexicales, ne s'applique que si la base ne termine pas l'occurrence et ouvre les dictionnaires 1, 2 et 3 ;
 - R2 traite les bases lexicales qui terminent l'occurrence ;
 - R3, identique à R1, contient la fonction -FINAL- ;
 - R4 traite les désinences

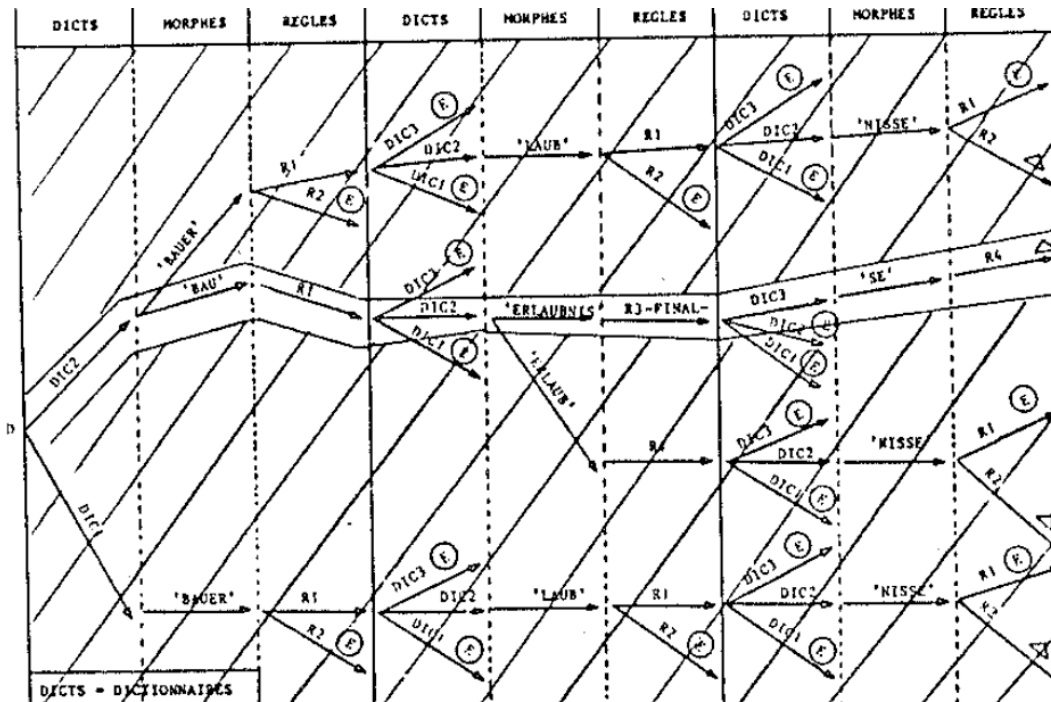


Figure 36: Arborecence de choix

D : Départ, E : Échec, Δ : État final

Le découpage obtenu est :
 BAUERLAUBNISSE = (BAU + ERLAUBNIS + SE)

Les règles peuvent comporter des conditions portant sur l'état courant (C), sur les chaînes C et A, sur les résultats partiels produits par l'analyse en cours (PS1 à PS9) en cas de mot composé, et aussi sur les quatre occurrences précédentes (de P1 à P4) et sur leurs résultats d'analyse. Une forme particulière de condition consiste à donner une liste de "sous-règles" et à exiger que l'une au moins s'applique (comme une sous-règle peut avoir des sous-règles, on travaille en mode non-déterministe unaire avec rétrogression). Il est enfin possible de mettre en réserve une condition sur l'analyse de l'occurrence suivante.

Il existe trois types d'actions : affectation de valeurs au masque C, transformation de ce qui reste à segmenter (chaîne A), et appel à des fonctions spéciales. Ces fonctions permettent de :

- contrôler la rétrogression (backtrack) incorporée, en élaguant l'arbre des choix (fonctions FINAL, ARRET, ARD, ARF, STOP) ou en ouvrant et fermant les dictionnaires (par affectation de la variable obligatoire DICT) ;
- produire un résultat partiel à partir de l'état courant C (fonction SOL) ;

- transformer C ou A en une UL, tout en réduisant A à la chaîne vide " (fonctions TRANS et TRANSA);
- décider qu'une borne de phrase est atteinte (fonction INIT).

En cas d'occurrence non reconnue (« mot inconnu »), c'est-à-dire si aucune analyse n'arrive à réduire A à " tout en produisant un état courant C muni d'une UL non vide, le système recommence l'analyse en attachant à l'occurrence le format morphologique obligatoire MODINC, qui doit en particulier appeler la règle obligatoire MOTINC, dite "règle du mot inconnu". Comme cette règle peut appeler des sous-règles et comme ce format peut appeler d'autres règles, on peut construire une vraie sous-grammaire du mot inconnu, et mettre en place des stratégies élaborées.

Au fur et à mesure du traitement, l'automate construit un graphe (quadrichrome arrière) dont les nœuds sont les masques (listes de masques pour les mots composés) associés aux solutions trouvées, et où les arcs indiquent la compatibilité des analyses entre elles. Le graphe final est ensuite transformé dans la forme souhaitée. Actuellement, la sortie en Q-graphes et la sortie en graphe monochrome avant ne sont plus disponibles. La sortie la plus utilisée est de forme « arbre sans homophrases », tandis que la sortie en arbre "avec homophrases" (tous les chemins du graphe quadrichrome sont présentés séparément) n'est utilisée que pour des systèmes comme LIDIA produisant des solutions multiples.

La sortie standard d'ATEF est donc un arbre "sans homophrases". La racine correspond à tout le texte, et porte UL='ULTXT'. Ses fils correspondent aux phrases (déterminées par la grammaire) et portent UL='ULFRA'. Sous chacun d'eux, on trouve les 'ULOCC' et 'ULTOURN' correspondant aux occurrences (mots ou tournures figées connexes). Sous chaque 'ULOCC', on trouve les différents résultats d'analyse morphologique de l'occurrence correspondante. Chaque résultat est, soit un masque de variables (un nœud), soit un sous-arbre de racine 'ULMCP' (mot composé) dominant les masques correspondant aux différentes parties reconnues dans ce mot.

Deux exemples de sortie d'ATEF sont donnés à l'annexe 1.1.

3.2.3.2.1.2.SYGMOR, un langage pour la génération morphologique

SYGMOR est fondé sur un modèle de transducteur d'états finis déterministe. Sa première version a été conçue par B. Thouin et programmée par D. Jaeger. [Guillaume, 1989a] présente les améliorations et extensions qu'il y a apportées par la suite. SYGMOR prend en entrée une suite de décorations et produit en sortie une chaîne de caractères. On dispose d'un contexte réduit aux décorations courante (C) et précédente (P), et on travaille sur deux chaînes, la chaîne de travail (T) et la chaîne en sortie (S).

La grammaire est de structure assez simple. Chaque règle comporte des conditions, des actions et des « règles suivantes ». Les actions consistent essentiellement à écrire à droite, à gauche ou au milieu (dernier point de concaténation) de T une chaîne littérale ou le résultat de la recherche dans l'un des dictionnaires à partir de la valeur de l'une des variables de C (les dictionnaires accédés par les UL donnent les bases, les autres les affixes). On peut aussi modifier C, et « rappeler » S dans T (en la concaténant à gauche et en l'effaçant).

Pour chaque décoration, SYGMOR recherche la première règle applicable, et l'applique. Il exécute alors dans l'ordre les règles suivantes, sans tenir compte de leurs parties « règles suivantes ». La liste peut comporter des règles facultatives. Si une règle obligatoire échoue, on revient à l'état initial et on applique, si elle est présente, la règle MOTINC, et sinon l'action d'erreur (vider S, puis S:=T). On continue en prenant en compte la partie « règles suivantes » de la dernière règle appliquée.

Un exemple de génération morphologique est donné en annexe 1.2.

3.2.3.2.2.Phase de transformation d'arbres décorés

3.2.3.2.2.1.TRACOMPL, pour la transformation de décorations (« articulations »)

TRACOMPL [Guillaume, 1989b] est le sous-langage d'écriture de tous les composants DV. On en a fait un langage autonome pour pouvoir écrire des "articulations". Il s'agit de transformer une décoration d'un Jeu1 vers un Jeu2. Pour cela, on procède en deux étapes :

- On décrit d'abord le Jeu2 et ce que l'on doit connaître du Jeu1 pour réaliser la transformation. Les noms des variables présentes dans les deux sont préfixés par "\$", et ceux des variables présentes dans Jeu1 et non reprises dans Jeu2 par "\$\$". Les autres sont considérées comme nouvelles.
- On complète cela en écrivant (partie CVAR) une action conditionnelle qui peut tester les variables de la décoration source en Jeu1, en Jeu2 après le "reformatage" décrit par la partie précédente, et en Jeu2 dans leur état courant. À l'aide de cette action, on peut par exemple transformer une variable à 2 valeurs en une variable à 3 valeurs, ou inversement.

Un exemple de transformation d'un arbre décoré est donné à l'annexe 1.3.

3.2.3.2.2.2.ROBRA, pour l'écriture de systèmes transformationnels d'arbres décorés

ROBRA [Boitet, Guillaume et Quézel-Ambrunaz, 1978] est un langage d'écriture de systèmes transformationnels agissant sur des arbres décorés. C'est le successeur du langage CETA [Chauché, 1975]. De nombreuses extensions y ont été apportées, la sémantique a été précisée dans certains cas, et le moteur a été totalement respécifié et réécrit (en 1977).

Un système transformationnel (ST) est défini par un *graphe de contrôle* (GC), un ensemble de *grammaires transformationnelles* (GT) et un ensemble de règles (RP pour « règles de production »). Une GT est un ensemble ordonné de règles. Un GC est un graphe dont les nœuds portent chacun une GT ou un symbole de sortie (&NUL) et dont les arcs portent des conditions d'arbres. Notons ici qu'un composant « grammaire » (GRi) d'une phase ROBRA contient en fait un système transformationnel, qui peut-être constitué d'un grand GC avec des dizaines de GT.

Pour exécuter un ST sur un *arbre objet* (AO), on utilise le GC comme structure de contrôle non déterministe (unaire avec rétrogression) : partant d'un nœud initial, on cherche le premier chemin licite menant à une sortie. Sur ce chemin, on exécute les grammaires contenues dans les nœuds, et, pour pouvoir traverser un arc, il faut que l'AO courant vérifie la condition qu'il porte.

L'exécution d'une GT est faite d'une application élémentaire en mode unitaire (U), ou de plusieurs en mode itératif (E pour « exhaustif »). Dans une application

élémentaire, on applique en parallèle le plus possible de règles de la GT, d'où un mécanisme de résolution de conflits. Une application élémentaire n'est terminée que lorsque les éventuels appels récursifs de sous-grammaires (SGT) ou de sous-systèmes (SST) provoqués par l'application de certaines règles de la GT sont terminés.

Un système d'interdictions (règles marquées, points bloqués) permet de tester statiquement la décidabilité du ST : le compilateur peut prévenir l'utilisateur des risques d'indécidabilité (boucle dans le GC, mode « libre » dans une GT itérative, contrainte de récursion non respectée, etc.).

Les schémas qui apparaissent en partie gauche de règles ont une très grande puissance d'expression. Pour chaque nœud, on peut indiquer si l'on recherche ses fils le plus à gauche ou le plus à droite possible, dans l'ordre ou dans le désordre. On peut rechercher des nœuds à des profondeurs inconnues à l'avance au moyen de "nœuds généralisés". Enfin, il s'agit de règles sous-contexte, la *racine du schéma (RS)* pouvant ne pas être confondue avec la *racine de la transformation effective (RT)*. Tout ce qui n'est pas dominé par la RT forme le contexte, ou « chapeau ». La RT peut être active ou contextuelle. Tout ce qu'elle domine est actif, sauf les listes.

La notion de réécriture parallèle en ROBRA est assez forte, puisqu'on peut avoir un parallélisme "normal" (RT disposées sur des nœuds distincts d'une coupe de l'AO), "vertical" (une RT peut en dominer une autre), et "horizontal" (plusieurs RT contextuelles et au plus une RT active peuvent s'instancier sur le même nœud de l'AO).

Enfin, on peut écrire en partie droite de règle des affectations conditionnelles des variables extrêmement complexes, ce qui contribue à faire de ROBRA un outil extrêmement puissant.

ROBRA est bien un modèle de substitution, même si, dans l'implémentation, l'exécution élémentaire d'une grammaire transformationnelle se fait par transduction d'un arbre en entrée vers un arbre en sortie. C'est pourquoi le type de décoration est nécessairement conservé.

Un exemple d'une transformation d'arbres en ROBRA est donné en annexe 1.4.

3.2.3.2.2.3.EXPANS/TRANSF, pour l'expansion lexicale monolingue & le transfert lexical bilingue

EXPANS [Guillaume, 1989b] est fondé sur un modèle de transduction d'arbres décorés. Les arbres d'entrée et de sortie peuvent être sur deux jeux de variables différents. Chaque nœud produit un sous-arbre dans l'arbre image. Ce sous-arbre est déterminé par consultation des dictionnaires, dans leur ordre de priorité, selon l'UL portée par le nœud. Une action par défaut est toujours prévue.

Un article de dictionnaire contient comme clef une valeur d'UL, et comme contenu une liste de triplets <condition,image,affectations>. Les conditions portent sur le nœud de l'arbre origine et éventuellement sur ses voisins immédiats (père, premier fils, frère droit). L'image exprime la géométrie du sous-arbre à produire, et les affectations permettent de calculer les valeurs des variables des nœuds du sous-arbre produit en fonction de celles des nœuds sources accessibles.

Au niveau d'un dictionnaire, si l'UL du nœud source est trouvée, on recherche le premier triplet dont la condition est vérifiée (la dernière condition doit être vide,

c'est-à-dire identiquement vraie), et on produit le sous-arbre correspondant. Sinon, il y a échec.

En mode déterministe, on parcourt les dictionnaires dans leur ordre de priorité jusqu'à obtenir un succès. En mode non-déterministe, on parcourt tous les dictionnaires, dans l'ordre, et on produit comme image un sous-arbre construit en enracinant les sous-arbres produits par les différents dictionnaires sous un nouveau nœud. Ce mode permet par exemple de ne pas « cacher » la traduction usuelle d'un mot qui a aussi une traduction différente dans un domaine particulier dont le dictionnaire serait prioritaire.

3.2.3.3.LSPL associés à Ariane-G5

3.2.3.3.1.ATLAS, pour l'aide à l'indexage dans les dictionnaires des LSPL

Il s'agit d'un langage d'écriture de « cartes d'indexage », spécifié et implémenté par D. Bachut [Bachut et Verastegui, 1984]. Il a été utilisé pour produire de nombreux manuels d'indexage du système russe-français. On décrit un graphe acyclique dont les nœuds internes portent des questions, les arcs les réponses possibles, et les feuilles les résultats auxquels on parvient (le plus souvent, des noms de formats ou de procédures). Un exemple d'indexage d'ATLAS est donné dans l'annexe 1.5.

Ce graphe peut être dessiné, pour fabriquer des manuels sur papier, ou être utilisé dynamiquement, pour créer des enchaînements de menus à l'écran dans une fenêtre, et envoyer les résultats aux endroits appropriés dans une seconde fenêtre où défile le dictionnaire à construire ou à compléter.

3.2.3.3.2.LT, un langage d'écriture de transcripteurs

LT [Yves Lepage, 1986] permet d'écrire rapidement des transcripteurs de textes. Par exemple, on utilise dans le système russe-français une transcription des textes russes dans un sous-ensemble du jeu de caractères de PL/I (majuscules, chiffres et quelques signes spéciaux). Un transcripteur écrit en LT permettait en 1990 d'obtenir les textes en cyrillique (non supporté par la configuration disponible), avec majuscules, minuscules et formatage minimal, sur une imprimante ASCII (S700) munie de fontes cyrilliques et connectée au 4361 via un PCI. LT a aussi été utilisé lors du projet LIDIA.

Le modèle abstrait est un transducteur d'états finis à deux bandes. La bande d'entrée est munie de deux têtes de lecture. La première ne peut qu'avancer, et la seconde, qui sert de "regard en avant", peut avancer ou être rappelée à la position de la première. La bande de sortie est munie d'une tête d'écriture. Les états sont structurés : un état complet est la combinaison d'un état élémentaire et d'un ensemble de valeurs d'un certain nombre de variables.

Écrire un transcripteur consiste à déclarer les variables et leurs types (par exemple, fonte, langue, longueur du regard en avant...), puis à décrire le graphe du système de transition, avec un nœud par état élémentaire. Les arcs portent les transitions, qui sont de classiques règles de production. Il y a un certain nombre de fonctions prédéfinies, ainsi que des possibilités de paramétrage et de factorisation, pour éviter d'avoir à écrire un trop grand nombre de règles. Un exemple de LT est donné à l'annexe 1.6.

La première version de LT a été définie et réalisée en PROLOG I par Ch. Boitet et un de ses étudiants au GETA durant l'hiver 82-83 dans le cadre d'un projet de DEA. Yves Lepage [Yves Lepage, 1989] a ensuite étendu la spécification et réalisé une seconde version

opérationnelle mais beaucoup trop lente. Il a ensuite réécrit le moteur en Pascal/VS et le compilateur en PROLOG-CRISS, ce qui a donné une troisième version encore très lente mais utilisable et utilisée. La quatrième version basée sur CLOS (Common Lisp Object System) de LT (LT4) a été implémentée par M. Lafourcade en 1994.

3.2.3.3.3. *TTEDIT, un éditeur transformationnel d'arbres décorés*

Pour développer indépendamment des analyseurs, des transferts et des générateurs, il faut disposer de jeux d'essai. [Durand, 1988] a développé pour cela TTEDIT, qui se présente comme un éditeur d'arbres (simplement étiquetés ou décorés), dont l'originalité est que ses opérations de base sont des transformations de sous-arbres, comme en ROBRA, et non des manipulations directes sur les nœuds ou les arcs. Cela permet de travailler sur de gros arbres comme on travaille sur des textes avec des éditeurs munis de fonctions de recherche et remplacement fortement paramétrées. TTEDIT est complètement intégré à l'éditeur standard de texte XEDIT. Comme en XEDIT, on peut écrire des "macros", mais il s'agit ici de grammaires transformationnelles analogues à celles de ROBRA.

[Guibaud, 1988] a utilisé TTEDIT pour, à partir d'un arbre produit par l'analyseur du français de B'VITAL et transformé par un transfert réalisé pour l'occasion (en Ariane), réaliser les transformations complémentaires *ad hoc* permettant de produire une structure interface d'analyse conforme à la "législation linguistique" du projet Eurotra et... aux sentiments personnels des responsables pour chaque cas particulier non prévu par cette "législation".

Un exemple de TTEDIT est donné dans l'annexe 1.7.

3.3. Composants linguiciels

3.3.1. Organisation des dictionnaires

3.3.1.1. Notions utilisées

Un dictionnaire associe des informations grammaticales et des références lexicales aux unités lexicales d'une langue.

Occurrences : une *forme* ou *occurrence* est une suite non vide de caractères, sans blanc, n'en contenant pas strictement une autre. En Ariane-G5, les caractères sont les 256 caractères EBCDIC, et la longueur d'une occurrence doit être comprise entre 1 et 255.

Morphes : un *morphe* est pour Ariane-G5 une suite de caractères ne comportant pas de blanc. Pour les linguistes, les morphes peuvent être des bases, des affixes, ou des sous-chaînes utiles de formes fléchies.

Unités lexicales (UL) : une *unité lexicale* (UL) est une valeur de la variable UL, prédéfinie et exclusive. Les unités lexicales sont introduites dans les dictionnaires et les grammaires, mais non comme liste déclarée de valeurs. Dans ce sens, la variable UL peut être dite « potentielle ».

La notion d'unité lexicale est très utile pour l'étape de génération. Elle permet de représenter de façon compacte des familles dérivationnelles. Les dictionnaires usuels modernes utilisent une notion analogue. En analyse, cette notion permet de diminuer la taille des dictionnaires, et surtout de pouvoir traiter de façon systématique les néologismes obtenus par des dérivations productives.

Par exemple, voici l'extraction du dictionnaire en SYGMOR de la phase GM du système FVX-ENX (français-anglais) :

(Flexions verbales)			
UL	Condition	Affectation	Chaîne
FLEX2	==	/	/.
	==TPRSS	/	/ES,
	==SVINGS	/	/ING,
	==SVINGSG	/	/ING'S,
	==SVINGPG	/	/INGS,
	==SVINGP	/	/INGS',
	==SVING	/	/ING,
	==SVVENG	/	/ED'S,
	==SVVENP	/	/EDS,
	==PRTPAST	/	/ED,
	==	/	/.
(Flexions vers le nom)			
FLEX3	==DVNASG	/	/IATION'S,
	==DVNAS	/	/IATION,
	==DVNAP	/	/IATIONS,
	==DVNAGASG	/	/IATOR'S,
	==DVNAGAS	/	/IATOR,
	==DVNAGAP	/	/IATORS,
	==DVNAISG	/	/YION'S,
	==DVNAIS	/	/YION
	==DVNAIP	/	/YION,
	==DVNAIGSG	/	/YIONS,

Du point de vue linguistique, on est naturellement conduit à regrouper, par exemple, *démarrer, démarreur, démarrage, démarrable,...* dans l'UL *démarrer-V*, les dérivations retenues ayant les 3 aspects sémantique, syntaxique et morphologique (ex: nom d'action en *-age*, nom d'agent ou d'instrument en *-eur*), dans l'ordre d'importance. Du point de vue pratique, on sépare souvent d'une telle famille le nom d'agent ou d'instrument, car la dérivation en question ne peut être utilisée pour produire des paraphrases correctes en traduction (« *démarrateur* » → « *ce qui fait démarrer* » ?), et car cela permet de séparer l'indexage purement terminologique de ces termes de l'indexage plus complexe de toute une famille de déverbaux.

Selon la classe syntaxique du lemme principal (source des dérivations), on distingue les UL verbales, nominales et adjectivales. Il y a bien sûr des UL réduites à un seul terme (mots outils, adverbes non dérivés comme *là, ici...*).

3.3.1.2.Espaces lexicaux

On appelle « espace lexical » d'une langue l'ensemble structuré de ses unités, à des niveaux de plus en plus abstraits ou génériques : formes, lemmes, racines (dans certaines langues) familles dérivationnelles plus ou moins productives, prolexèmes (réunissant par exemple « US », « USA » et « Etats-Unis »), et « acception » ou « sens de mots »... sans oublier qu'il y a à tous ces niveaux des unités simples et des unités complexes (mots composés, lexies ou acceptions complexes).

3.3.1.2.1.Organisation

En Ariane-G5, l'espace lexical d'une langue (source et cible) repérée par un « code langue » (RUS, FRA, FR1, ENG, EN3...) contient les occurrences (conservées dans les décorations, mais non manipulables) et ce que les linguistes décident de représenter à l'aide de la variable UL, et éventuellement des variables supplémentaires (déclarées) comme le « numéro de sens ». Dans plusieurs systèmes écrit en Ariane-G5, la combinaison UL+numéro de sens+CAT (CAT pour la classe morphosyntaxique, POS en anglais) correspond à une « lexie » (sens d'un lemme simple ou composé dans un dictionnaire usuel).

Pour permettre la compilation séparée, chaque composant linguiciel compilé (grammaire ou dictionnaire ou procédures ou formats) est accompagné de la liste triée de ses UL et de ces occurrences.

Au moment où on "charge" un linguiciel en mémoire, juste avant une exécution, ou bien pour préparer un « load module » (déjà tout prêt), on remplace dans chaque composant linguiciel compilé les numéros internes d'UL locaux par les numéros globaux (ceux de "l'espace lexical source" ou de "l'espace lexical cible") dans toutes les décorations et dans les codes compilés (par exemple, des conditions et affectations). C'est une opération tout à fait similaire à une édition de liens (link edit) en informatique classique.

3.3.1.2.2. Traitement des UL dynamiques

Ariane-G5 maintient une liste triée de toutes les UL statiques (apparaissant dans les composants linguiciels) "source" ou "cible" d'un "code langue source" ou d'un "code langue cible".

Toutes les UL sont référencées par un identificateur. Si une UL n'existe pas dans un dictionnaire ou une procédure (par exemple dans une AM sous ATEF), une règle peut demander de créer une UL non-existante. On lui attribue temporairement un numéro négatif libre à partir de -7 (de -1 à -6 réservés pour les valeurs prédéclarées 'ULTXT', 'ULFRA', 'ULSOL', 'ULTOURN', 'ULOCC', 'ULMCP'). Elle est considérée comme « dynamique ».

En phase de transfert, une procédure de vérification de la cohérence des UL est réalisée selon le contenu d'UL. Si l'on trouve une UL dans un dictionnaire ouvert correspondant à cette UL, on lui attache son numéro. Si non, on passe dans le dictionnaire D0 où il y a un article par défaut qui traite cette UL comme un mot inconnu.

L'ensemble des UL dynamiques est remis à vide au début du traitement de chaque unité de traduction.

Au niveau de la gestion, on établit bien sûr une liste des UL négatives apparues pour chaque segment, pour que les développeurs puissent compléter les dictionnaires s'ils le souhaitent.

3.3.1.2.3. Représentations usuelles des « acceptions » (ou « sens de mots »)

Lors de la conception d'Ariane-78 et d'Ariane-G5, on n'avait pas encore la notion d'acception (sens de mot « en usage »). Pour la représenter, on a souvent utilisé pour un variable NUMSENS arithmétique, par exemple à 255 valeurs pour indiquer implicitement le sens de cette UL. En correspondance à chaque valeur, on définit alors des propriétaires syntaxiques sous forme attribut/valeur pour détailler ce sens (ex. verbe transitif ou non transitif...).

3.3.1.3. Fonctions utilitaires

Tous les dictionnaires d'une phase sont écrits dans le LSPL associé à cette phase. Donc, on peut faire l'indexage, vérifier la syntaxe, trier ou faire des vérifications croisées (par exemple, trouver que telle UL est utilisée dans quelles applications, quelles phases, quelles langues).

3.3.1.3.1. Indexage

L'indexage est fait directement sous XEDIT, ou en utilisant l'outil ATLAS (voir 3.2.3.3.1), ou encore, depuis un Mac ou un PC, en utilisant les aides à l'indexage de CASH. Quand l'utilisateur ajoute une entrée dans le dictionnaire, il cherche à saisir le code associé pour cette UL. Grâce à ATLAS, on peut le définir en suivant une « carte d'indexage » sous forme de menus de questions/réponses. Enfin, on interprète la carte pour avoir la définition correspondante.

3.3.1.3.2. Tris & listes

Pour lister le dictionnaire d'une phase, on a les commandes DIC (pour la visualisation, modification ou compilation d'un dictionnaire), LISDIC (lister les dictionnaires), LISNUMAUXI ou LISAUXI (lister les dictionnaires auxiliaires), LISCOM (lister les dictionnaires complémentaires). Ces commandes possèdent un paramètre pour trier la liste

résultat par ordre alphabétique de gauche à droite selon le morphe, le format morphologique ou le format syntaxique de chaque article.

3.3.1.3.3. Vérification individuelle et vérification « croisée »

Quand on compile un dictionnaire, le compilateur du LSPL (ATEF, EXPANS/TRANSF ou SYGMOR) vérifie syntaxiquement la correction et la cohérence des articles individuels écrits dans une phase, et entre phases quand on construit une chaîne d'exécution. On peut également demander de sortir des « références croisées » des articles d'un dictionnaire lors de la compilation. Par exemple dans la compilation du dictionnaire numéro 1 sous TRANSF:

```
DIC 1 CTS OUI ; Vérifier le dictionnaire numéro 1, puis imprimer la
sortie à l'écran.
```

3.3.2. Traitement des corpus

3.3.2.1. Conventions de nommage

Comme le système d'exploitation utilisé (CMS) n'avait pas de notion de répertoire hiérarchique quand Ariane-G5 a été conçu, les fichiers contenant les composants linguiciels (source et compilés) sont « à plat ». D'autre part, un fichier, est identifié par un nom et un type. On a donc utilisé des conventions de nommage de noms logiques à noms physiques de VM/CMS. Un corpus a un nom, et ses textes sont dans des fichiers dont le nom est celui du corpus et le type un identificateur de texte quelconque. Dans le projet russe-français (1982-1987), on avait des conventions supplémentaires pour identifier les textes. Par exemple, DRET C0302002 représentait le texte numéro 002 dans le corpus nommé DRET, reçu le 03 février 1984 (C signifiait la 3^{ème} année du contrat, soit 1984).

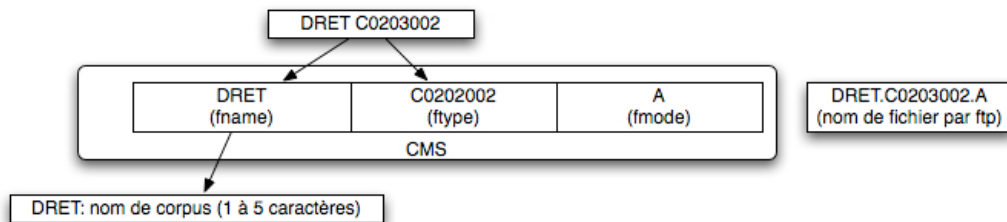


Figure 37: Exemple de la convention de nommage de corpus du système Ariane-G5

3.3.2.2. Organisation des fichiers pour une traduction

Sur une traduction d'un fichier texte, on obtient en organisation interne :

- fichier texte source.
- fichier de description arborescente (utilisé pour calculer les unités de traduction).
- fichier de segmentation en unités de traduction selon une fenêtre paramétrable par le linguiste (200-300 caractères).
- fichiers d'arbre de résultats intermédiaires possibles.
 - chaque fichier selon la chaîne d'exécution/production
 - fichiers selon les phrases. Chaque fichier contient la liste des arbres fournis par cette phase (sauf GM en SYGMOR, où les résultats sont des chaînes générées de la langue cible), un pour chaque unité de traduction.
- fichiers résultat de traduction automatique d'un ou plusieurs chaînes d'exécution. Le numéro de la chaîne d'exécution est inclus dans le nom de fichier.
- fichiers de révision.

- fichiers de résultat formaté.

Avec un corpus, on a 2 fichiers « spéciaux », \$DESCR contenant la liste des textes de ce corpus avec le format <nom externe, nom interne>, et un autre contenant une liste hiérarchique des séparateurs.

3.3.2.3.Types de fichiers de résultat

3.3.2.3.1.Résultat intermédiaire (liste d'arbres décorés)

Le résultat intermédiaire est le format d'échange entre les étapes du système. Ce format représente des arbres décorés, les formes et UL utilisées (Figure 38).

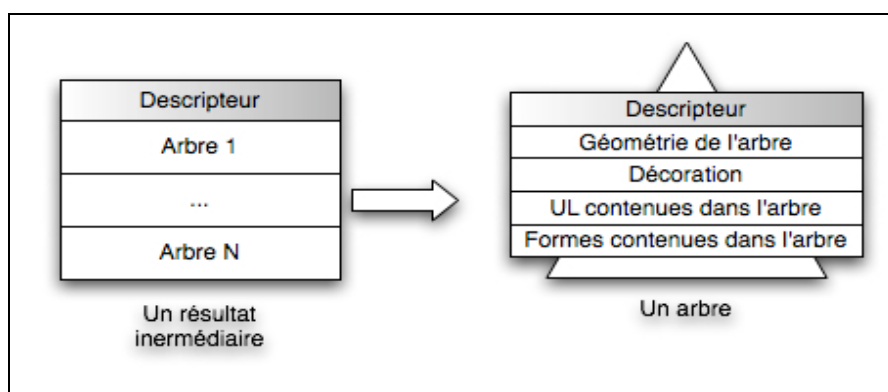


Figure 38: Format de résultat intermédiaire de traduction

Quand un arbre est soumis à une « phase », on le « charge » en remplaçant dans chaque décoration le numéro interne (sur 2 octets) de l'UL par le numéro de cette UL dans l'ensemble de toutes les UL connues (apparaissant dans les composants linguiciels ou dans l'arbre) trié lexicographiquement.

3.3.2.3.2.Traduction brute

Le résultat brut de la traduction est la concaténation des traductions des unités de traduction, et on n'a plus accès aux unités individuelles. Pour intégrer cela à un système à « petits segments » comme SECTra_w, il faut donc a posteriori resegmenter et réaligner les segments source et cible. Mais, comme les unités de traduction d'Ariane-G5 sont en général un ou plusieurs paragraphes, voire pages, conserver cette division n'aiderait que faiblement la segmentation et l'alignement à une granularité plus fine.

3.3.2.3.3.Révision

Pour un texte donné, il y a un seul fichier de révision par langue et par chaîne de traduction. Il n'y a pas de gestion de versions: la dernière modification est prise en compte. Il n'y a pas non plus de réinsertion des éléments hors-texte, pour la bonne raison qu'on n'avait pas à l'époque de moyens de saisir en machine des images, des formules chimiques, ou même des formules mathématiques.

Rien n'empêcherait de compléter cela, mais on préfère le faire dans le projet Ariane-Y.

3.3.3. Programmation linguistique

3.3.3.1. Méthodologie directe

Dans certains systèmes de TA comme SYSTRAN, les linguiciels sont codés directement dans un langage de programmation³⁷. Pour chaque composant, la programmation linguistique est un mélange entre la définition linguistique, la programmation informatique et la manipulation pour compiler, exécuter.

3.3.3.2. Grammaires statiques pour la spécification semi-formelle

Le niveau supérieur de la méthodologie est d'utiliser les grammaires statiques. Dans les étapes d'analyse ou de génération, on spécifie la correspondance <entrée, sortie> par des règles statiques.

3.3.3.3. Stratégie d'utilisation des LSPL

Voici deux exemples d'utilisation typique d'ATEF et ROBRA dans plusieurs systèmes de TA écrits en Ariane-G5.

3.3.3.3.1. ATEF

Dans l'analyse morphologique³⁸, on distingue entre mots invariables (base) et mots variables (acceptation des désinences); à constater pour ces derniers les différents modes de conjugaison (verbes) ou de déclinaison (noms, pronoms et adjectifs). Les classes sont définies par conséquent surtout à l'aide de valeurs de variables stratégiques. Et, pour la définition des classes de désinences, on a souvent recours aux catégories de MODE, TEMPS, PERSONNE, CAS, NOMBRE, *etc.* par rapport à chacune des valeurs de variables stratégiques : telle classe de désinences renseigne sur l'actualisation de telle classe de lexèmes. Les classes de désinences sont ce qu'on appelle parfois les morphèmes grammaticaux, leur extension est donnée par un morphe ou une série d'allomorphes.

Pour ce faire, on examine « a priori » toutes les analyses possibles dans le découpage de l'occurrence à analyser. Chaque étape d'une analyse particulière consiste à découper un segment dans « ce qui reste » de la forme à analyser et à appliquer l'une des règles référencées par le format "morphologique" associé à ce segment sur l'article correspondant relevé dans le dictionnaire. Les dictionnaires contiennent toutes les chaînes élémentaires intervenant lors du découpage des occurrences d'un texte par l'automate.

Le système permet l'utilisation de 1 à 14 dictionnaires, dictionnaires de tournures inclus. On a donc la possibilité de créer plusieurs dictionnaires pour la répartition des bases et des désinences. Cette possibilité jointe à l'utilisation des différentes FONCTIONS dans les règles de grammaire permet de guider la consultation des dictionnaires, en vue d'accélérer le découpage et l'interprétation des occurrences et d'éviter les résultats parasites.

Pour la segmentation d'une occurrence, les dictionnaires validés par la règle

³⁷ En SYSTRAN, l'analyse morphologique est effectuée par transducteurs finis, donnant en sortie un graphe de possibilités, tandis que l'analyse morphologique est écrite en « instanciant » un programme C générique, grâce à des « macro » hérité de la première implémentation en macro Assembler 360, et correspondant à un modèle déterministe produisant toujours une solution unique.

³⁸ Dans cette partie, nous nous sommes essentiellement inspirés de la thèse de Jean-Philippe Guilbaud.

précédente sont consultés séquentiellement et en commençant par le dictionnaire de numéro le plus petit. A chaque fois, les morphes candidats sont empilés, du plus court au plus long. Si dans un même dictionnaire figurent plusieurs morphes identiques (de même longueur), ils sont empilés dans l'ordre d'apparition de leurs noms dans la grammaire, celle-ci étant lue de haut en bas et ensuite selon l'ordre alphabétique de leurs formats FTS. Si la segmentation d'une occurrence s'effectue en plus d'un pas, la même opération est systématiquement répétée avec les nouveaux dictionnaires validés par la règle qui a assuré la transition vers l'état suivant, que l'indication du ou des numéros de dictionnaire résulte de conventions qui rendent leur validation implicite, ou qu'elle soit faite de façon explicite dans la règle de grammaire.

En cas de forme non reconnue (« mot inconnu »), le système recommence l'analyse en rattachant la forme au format spécial (obligatoire) MODINC, qui doit en particulier appeler la règle obligatoire, dite « règle du mot inconnu », MOTINC. Comme cette règle peut appeler des sous-règles et que ce format peut appeler d'autres règles, la porte est ainsi ouverte à des stratégies élaborées. De plus, en mode interactif, le système s'arrête quand il rencontre un mot inconnu et demande à l'utilisateur s'il veut changer l'entrée (par exemple, dans le cas d'une faute d'orthographe) ou indexer de nouveaux articles (provisoires) dans les dictionnaires dits « auxiliaires ».

3.3.3.3.2. **ROBRA**

En ROBRA³⁹, le type de programmation est très différent selon qu'il s'agit d'analyse, de transfert ou de génération. En analyse, on commence par travailler en parallèle sur toute l'unité de traduction pour procéder à une normalisation de l'arbre (mots composés, ambiguïtés immédiatement solubles, regroupement de tournures prédites et trouvées, dates, noms propres, *etc.*). Ensuite, on provoque un appel récursif d'un sous-système transformationnel sur chaque phrase, puis éventuellement sur chaque sous-groupe sûr et assez gros, pour que la stratégie (parcours du graphe de contrôle) soit effectivement « dirigée par les données ». À la fin de l'analyse, on revient à un traitement de tout l'arbre, ce qui permet par exemple de tenter de trouver les référents des pronoms pour lesquels on n'en a pas trouvé à l'intérieur de la phrase où ils apparaissent.

En transfert, c'est assez simple. On traite les traductions de groupes complexes, on vérifie éventuellement les conditions contextuelles codées dans des sous-arbres représentant des traductions multiples, pour réduire les polysémies, puis on ajuste la g-structure, en y codant éventuellement des conseils ou des ordres au générateur en vue de produire telle ou telle forme syntaxique.

En génération, on travaille en descente récursive, pour produire la m-structure, puis pour commencer à construire la c-structure. On termine en général en traitant en parallèle l'affectation finale des variables d'actualisation de surface (accords, concordances), ce qui peut amener à modifier légèrement la géométrie (insertion d'auxiliaires avec traitement des clitiques). Il faut faire attention à ne pas utiliser de grammaires à la fois récursives et itératives, ce qui provoque de nombreux et coûteux appels de grammaires inutiles.

³⁹ Dans cette partie, nous nous sommes essentiellement inspirés de la DSE-1 rédigée par Christian Boitet.

3.4. Le projet Ariane-Y et l'intégration de LSPL variés (Ariane-Y complet)

Le système Ariane-G5 est un système considérable dans le domaine de traduction assistée par ordinateur. Cependant, la multilinguïisation et l'accès de l'information prennent de plus en plus une grande importance dans une époque où la technologie de communication et de calcul évoluent rapidement.

Dans cette situation, le système Ariane-G5 rencontre des difficultés comme la portabilité (bien qu'on ait déjà des méta-EDL, simulateur d'IBM comme TN 3270X à distance, on voudrait faire tourner des moteurs d'Ariane sur Unix/Linux et l'accès graphique sur plusieurs plates-formes : Unix, Linux, Windows, Mac...), l'interaction homme-machine pour développer des ressources à distance sous un environnement plus moderne comme le Web, l'intégration avec d'autres modules pour certains traitements supplémentaires, ... ainsi que les limites d'implémentation provenant des limites des moyens techniques au moment où Ariane-G5 a été conçu et développé.

Pour ces raisons, il est intéressant de faire la réingénierie du système Ariane-G5 vers un système plus moderne, plus portable, plus ouvert. C'est le projet Ariane-Y.

Pour ce faire, on est parti du niveau du système, pas du niveau du code source. Ce choix est justifié par les ressources limitées (certains développeurs d'Ariane-G5 sont partis en retraite et c'est un projet de recherche de GETALP avec l'étude et l'expérience de conception et de développement dans le domaine de TA, TAO), et l'on peut profiter des programmes d'Ariane-G5 qui fonctionnent très bien. On monte au niveau d'architecture du système, de l'intégration et de l'extension en supprimant les limites d'implémentation d'Ariane-G5.

La première idée du projet Ariane-Y a été formulé par Ch. Boitet en 1998, En 1999, on a défini le projet Ariane-Y comme un projet de recherche pour la demande d'AFIP de J-C. Durand, en 2000. Pour un tel projet ambitieux, il faut de grandes ressources (Ariane-G5 avait coûté 40 hxa, et LIDIA 10 hxa de plus, soit 50 hxa). On a écrit certaines documentations comme CCH, DSE d'architecture logicielle, DSE de langages, convention de programmation, ...

Actuellement, on a avancé sur certains développements et sur la conception de la construction du système Ariane-Y vers un système hétérogène. On a enlevé des limites à surmonter et des principes retenus dans la stratégie de la réalisation.

3.4.1. Limites d'Ariane-G5 à lever

3.4.1.1. Limites d'implémentation

Ariane-G5 traite actuellement une unité de traduction qui peut être une phrase, un paragraphe ou un document de quelques pages. On voudrait pouvoir traduire de très gros document (400 pages) comme une seule unité de traduction. C'est ce que peut d'ailleurs déjà faire l'analyseur XIP (Xerox Incremental Parsing) de Xerox [Aït-Mokhtar, Chanod et Roux, 2002].

3.4.1.1.1. Compilation non incrémentale

Actuellement, quand on ajoute ou modifie un objet linguiciel (un article de dictionnaire, une règle de grammaire...), on doit recompiler tout ce dictionnaire ou cette grammaire pour pouvoir effectuer la traduction. On voudrait dans Ariane-Y pouvoir utiliser de la compilation incrémentale pour cela. Cette fonction est encore plus importante quand on ouvre le système vers un système hétérogène. Par exemple, on veut pouvoir accéder à une base lexicale lors de l'exécution, quand l'AM (ou le TL) rencontre un mot (ou une UL) inconnu(e).

3.4.1.1.2. Accessibilité par réseau mais non-portabilité

Pour des raisons historiques, puis de moyens, Ariane-G5 tourne sur des machines IBM d'architecture 390 (actuellement un z/VM du GETALP), même s'il est accessible par le réseau (via TN 3270X, ou via les méta-EDL CASH et WICALE). On désire avoir un système

fonctionnant sur des plates-formes communes comme Unix/Linux, pour qu'on puisse bénéficier des acquis scientifiques d'autres groupes et projets du laboratoire, ainsi que du secteur industriel. L'installation d'Ariane sur les plates-formes communes permet d'intégrer facilement des modules et outils supplémentaires, y compris l'interface graphique qu'on n'avait pas avant.

Enfin, l'architecture actuelle ne se prête pas à la réalisation de systèmes de « TAO embarquée » sur un PDA ou un smartphone.

3.4.1.1.3. Diversité des langages de programmation

Le logiciel de base complet représente environ 500 000 lignes source en divers langages (ASM370, PL360 (moteur ROBRA avec certains modules supplémentaires d'ATEF), Pascal (ATLAS, VISULEX)), PL/I (compilation des dictionnaires et grammaires de SYGMOR), et surtout EXEC/REXX).

Cela est dû à des raisons historiques : absence de langages de haut niveau efficaces, et facturation du temps de calcul jusqu'à ce que le GETA dispose d'une machine dédiée.

3.4.1.1.4. Architecture du processus de traduction pas assez générique

Suite à l'expérience d'utilisation d'Ariane-G5, on observe que l'ajout de phases facultatives (voir 3.1.2.1) dans la chaîne de traduction d'Ariane-78 est un peu trop compliqué, même s'il est dû à une demande des développeurs (au GETA pour le russe-français et l'analyse des verbes à particule séparable en allemand, et à B' Vital pour des variantes du système français-anglais pour manuels techniques).

Notons qu'il existe au niveau interne d'Ariane-G5 un langage qui permet de décrire les séquences de phases (et d'articulations) d'Ariane-G5 (LIDS). Il nous faudrait le reprendre et l'utiliser pour décrire les « chaînes d'exécution » avec tous leurs paramètres.

3.4.1.2. Limitation à certains LSPL

3.4.1.2.1. Désir naturel d'intégrer d'autres LSPL

Ariane-G5 supporte 5 LSPL (voir 3.2.3.2). Or, on souhaiterait pouvoir en intégrer d'autres. Par exemple, pour le projet LIDIA, on aurait préféré écrire l'analyseur multiple avec un langage de type ATN (ou REZO du projet TAUM-aviation). On aurait aussi souhaité intégrer LT (langage de transcripteurs) pour que les linguistes écrivent les prétraitements.

Enfin, dès le tout début d'Ariane, en 1970-1971, on souhaitait y intégrer les systèmes-Q, créés à Montréal par A. Colmerauer en 1967, à côté des LSPL créés à Grenoble. Cela ne fut finalement pas fait, car les structures de représentation des énoncés étaient trop différentes : graphes d'arbres simplement étiquetés pour les systèmes-Q, et arbres à décorations complexes et typées pour les LSPL du GETA.

On voit là que ce désir d'intégrer un grand nombre de LSPL se heurte à un problème important, qui est d'assurer l'interopérabilité des structures de données.

3.4.1.2.2. Absence de phases à architecture computationnelle empirique

Bien que le système Ariane soit basé sur les connaissances linguistiques, on voudrait y ajouter des traitements utilisant l'approche empirique sur quelques tâches. Par exemple, à partir des gros corpus d'arbres de résultats intermédiaires, on devrait pouvoir créer une application de TA avec transfert « empirique » pour un domaine, en s'inspirant de l'approche de Microsoft en 2000-01.

3.4.1.3. Mise en réseau complexe

3.4.1.3.1. Qualité de l'idée de base

L'idée de l'extensibilité de service d'Ariane par le réseau LIDIA reste bonne. Cela ne change pas l'architecture logicielle du système et il y a une indépendance entre l'interface de communication (qui rend le service) et la partie centrale de traitement (TA, TAO), ...

3.4.1.3.2. Implémentation complexe car en avance sur le Web

Cependant, la technique d'implémentation est complexe car on était en avance sur le Web à cette époque. P. Guillaume a bien développé l'échange de données et commandes entre les machines via les spools de VM. Si on voulait porter le système Ariane sur plusieurs plates-formes, il nous faudrait une technique moins dépendante des plates-formes.

3.4.1.3.3. Esquisse d'une solution : des machines virtuelles vers des serveurs et/ou agents

La solution retenue est qu'on transforme l'architecture utilisant les machines virtuelles vers une architecture à serveur/services et agents étant entendu que, sous VM, les machines virtuelles restent de bon candidats pour la réalisation d'agent à gros gains, qui pourrait être réalisé autrement sur d'autres systèmes (Windows, Linux) n'ayant pas ce niveau de virtualisation. On construit un service de réception et interprétation des commandes depuis les moniteurs à distance. Le portage est alors réalisé par ce service « pivot » qui opère comme un serveur de communication entre le noyau du système Ariane vers l'extérieur, via les mécanismes de « pile-file » de REXX (REXX stack).

On prévoit également d'utiliser un format d'échange à la XML pour faciliter la technologie d'implémentation. En plus, on peut considérer plusieurs systèmes à la fois reliés avec les flot de données (à la XML). Toutes les commandes et données sont mises dans ce flot, et un système fonctionne comme un agent à gros gain. Cette architecture a été d'ailleurs mise en œuvre par Systran [Senellart, Boitet et Romary, 2003].

3.4.2. Grandes stratégies retenues

L'objectif pour Ariane-Y est triple : porter, unifier et étendre. Pour y parvenir, on suit 3 idées principales : simplifier pour généraliser, étendre tout en partant, et ouvrir à la TAO hétérogène.

3.4.2.1. Simplifier pour généraliser

Au niveau du moniteur, le passage d'une architecture à machines virtuelles vers une architecture à serveurs/services nous permet d'étendre l'accessibilité au système en intervenant sur la couche supérieure, sans modifier les programmes du réseau LIDIA.

Au niveau des LSPL, on unifiera encore plus qu'en Ariane-G5 tous les composants unifiables (compilateurs de composants analogues de langages spécialisés différents) ainsi que toutes les interfaces (arriver à une API unifiée pour les 5 langages spécialisés).

On supprimerait les phrases facultatives et on les remplacerait par la notion de « modèle » ou « prototype » de chaîne d'exécution. Pour ce faire, on aura un « superviseur » qui enchaînera les actions, les moteurs et linguiciels compilés étant en mémoire. Enfin, on utilisera des piles-files REXX pour communiquer.

3.4.2.2. Portabilité et extensibilité

On réalise le portage du logiciel Ariane-G5 sur une ou plusieurs plates-formes logicielles et matérielles conformes aux standards actuels, tout en levant les limites d'implémentation actuelles (nombre de variables, taille des dictionnaires).

Pour l'environnement à distance, on a aussi toutes les briques de base sur Mac ou PC, et on pourra reconstruire un environnement de façon plus moderne, par exemple en Java (WICALE), Revolution (CASH), ...

3.4.2.3. Ouverture à la TAO hétérogène

En utilisant des bases lexicales (PIVAX) et corporales (SECTra_w) externes, on partagerait le traitement et données avec autres systèmes de THAM et l'on déléguerait la partie de gestion des ressources lexicales et corporales à des systèmes associés adéquats conçus pour ces fonctions.

Pour mettre en service l'interaction entre composants d'Ariane et l'ouverture vers d'autres systèmes, on définit un format intermédiaire des représentations linguistiques des unités de traduction en XML (AY-XML) afin de faciliter l'échange. On prévoit une architecture de flot de données où chaque composant participe comme un agent et contribue au flot XML pour une des étapes de traitement dans le système Ariane-Y hétérogène.

Au niveau du moniteur, nous sommes entrain de construire un moniteur unique intégrant des commandes et des données du système. Ce moniteur simplifie l'échange et généralise l'accessibilité. C'est l'étape nécessaire pour faire évoluer l'architecture du système vers une architecture à service et agents.

3.4.3. Architecture générale

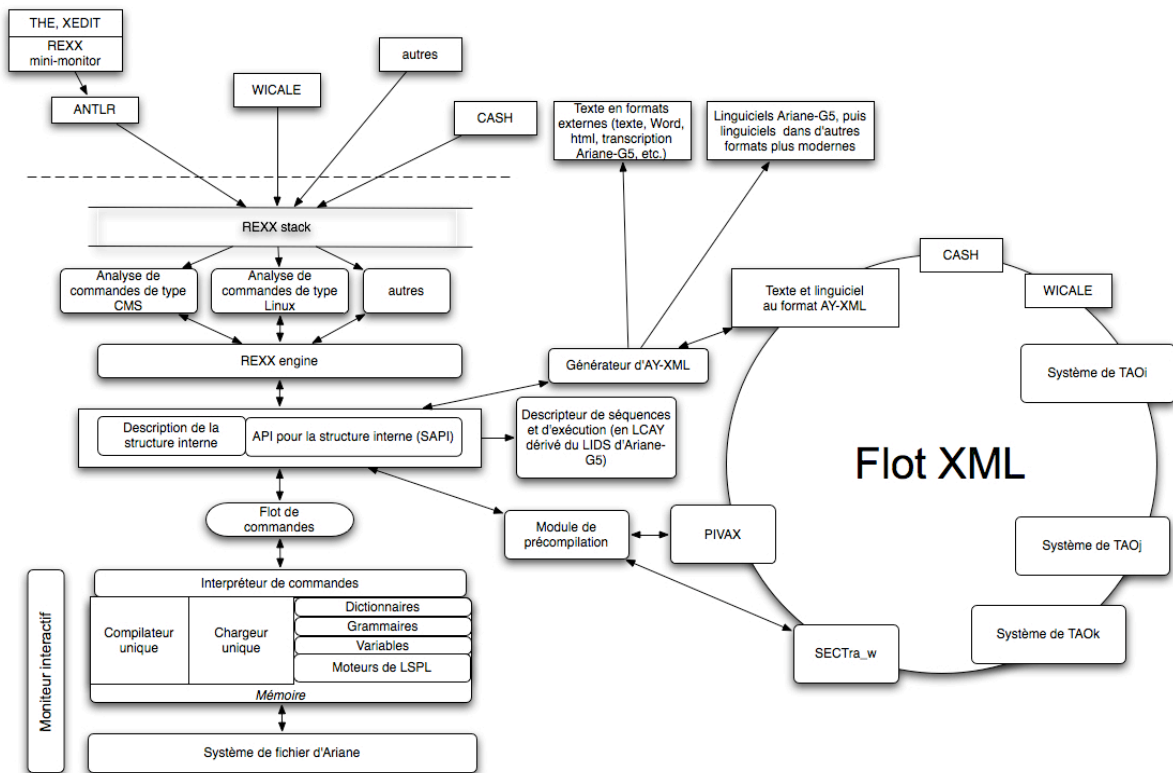


Figure 39: Schéma général de l'architecture logicielle d'Ariane-Y

Conclusion

Dans la première partie, on a montré l'évolution des systèmes de TA et THAM vers des systèmes de TAO hétérogènes et collaboratifs. On a vu également les principes de construction d'un grand système de TA en cours de réingénierie, le système Ariane-G5 et le projet Ariane-Y. Cet état de l'art justifie l'idée principale de la thèse : trouver comment construire des systèmes de TAO hétérogènes à partir de composants de systèmes existants.

Les problèmes « durs » proviennent de plusieurs niveaux, linguiciel, logiciel et opérationnel. En linguiciel, certains ont été étudiés et au moins partiellement résolus dans le cadre de cette thèse et de celle en cours de Cong-Phap HUYNH, comme la connaissance lexicale (base

lexicale universelle avec PIVAX), le traitement de corpus de traduction (base corporale en cours avec SECTra_w), mais d'autres restent encore insurmontés, comme la construction d'une base grammaticale commune ou d'une base lexicale pouvant accueillir et surtout relier tous les dictionnaires de TA connus.

En logiciel, l'aspect architectural d'un méta-EDL universel a été abordé depuis mon M2R au GETALP. Cette étude a conduit à des systèmes réels comme WICALE et EMEU_w. Cependant, si l'on veut qu'un méta-EDL soit plus pratique qu'un EDL, il nous faut le spécifier et demander d'importer certains traitements d'un ou de plusieurs EDL distants vers le méta-EDL.

Cette transformation nous mène à l'étude de la construction des EDL, et en particulier à celle de la réalisation de LSPL, qui permet de dégager l'approche la plus adéquate (compilateur et « moteur »), et les principes de base du génie logiciel pour le génie linguiciel. L'implémentation d'un LSPL validera ces principes.

Au niveau opérationnel, l'étude et la réalisation dans le cadre de la construction d'un système hétérogène sur le Web, dans le cadre du projet iMAG, fera émerger d'autres problèmes intéressants.

L'ensemble des études sur de tels aspects nous donnera une vue globale sur la construction des systèmes de TAO hétérogènes à partir de systèmes de TA homogènes existants.

Partie 2. Méthodes pour la convergence TA ↔ THAM

Introduction

La partie 1 a clarifié les notions de système de Traduction Automatique (TA) homogène, de Traduction Humaine Assistée par la Machine (THAM), et de Traduction Assistée par l'Ordinateur (TAO) hétérogène, grâce à une analyse de l'existant et un exemple très détaillé (Ariane-G5). Dans cette partie, nous présentons en détail les principes de réalisation de la convergence entre un système de TA et un système de THAM pour arriver à un système de TAO hétérogène, selon tous les aspects principaux : logiciel, linguiciel et la mise en œuvre.

Cette partie est divisée en 3 chapitres. Le chapitre 4 présente les aspects concernant les composants logiciel y compris : l'Environnement de Développement Linguiciel (EDL), les Langages Spécialisés pour la Programmation Linguiciel (LSPL) et le moniteur. Le chapitre 5 traite des parties linguicielles : dictionnaires, corpus et grammaires. Le dernier chapitre présente des aspects liés à l'application des techniques provenant de la THAM dans un système de TA.

Chapitre 4. Intervention de non-spécialistes sur les composants logiciels de la TA

Introduction

Actuellement, les utilisateurs de TH n'utilisent la TA que via des appels à des serveurs de TA pour prétraduire des documents, puis ils post-éditent la sortie. Un niveau un peu plus avancé est d'ajouter des dictionnaires utilisateur pour spécialiser la TA sur des domaines. Cette limite freine l'utilité de la TA pour la TH.

Le développement d'un système de TA est toujours réservé à des développeurs informaticiens et linguistes. Un utilisateur non-spécialisé ne peut pas intervenir sur les composants de TA et définir lui-même des fonctions et tâches selon ses besoins. Notre étude vise à briser cette séparation.

Un premier effort est d'offrir en local des fonctions de développement des systèmes de TA distants. Il s'agit en fait d'une délocalisation des fonctions. On l'a bien vu dans des gros systèmes de TA : CASH pour Ariane-G5, un logiciel léger en local pour se connecter à un serveur de SYSTRAN... La création d'un tel méta-EDL dédié à chaque système demande des implémentations spécifiques. On voudrait automatiser totalement ou partiellement cette construction en paramétrant des commandes et des données d'échange. On a expérimenté cette approche en étendant WICALE, un premier méta-EDL créé par V.Carpéna en 2004.

Une autre approche est de permettre aux utilisateurs de définir eux-mêmes des fonctions et des tâches par un langage spécialisé. On a bien vu son efficacité dans les systèmes Ariane-G5 (5 LSPL), MÉTÉO (Systèmes-Q), MU (GRADE)... Quand le système supporte des utilisateurs non-spécialisés, il nous faut faire la réingénierie des LSPL en vue de supporter différents types d'utilisateurs. L'implémentation d'un LSPL réel comme les systèmes-Q nous aide à expérimenter cette approche.

Pour bien fournir les fonctions aux utilisateurs, les moniteurs des systèmes de TA doivent être intuitifs et souples. Leur réalisation technique est un aspect important qu'on doit étudier.

Dans ce chapitre, on étudie systématiquement ces trois grands aspects : EDL, LSPL et moniteur.

4.1 Aspects liés aux EDL

Notre étude des EDL et méta-EDL a été menée en 2005 [Nguyen, 2005], suite de l'idée de D. Genthial implémentée dans [Carpena, 2004].

La motivation de la création d'un méta-EDL vient de la volonté de travailler à distance d'un EDL central. La fonctionnalité principale d'un méta-EDL est de donner accès aux données et aux commandes d'un EDL distant sur une machine locale. En génie logiciel, un méta-EDL offre tous les avantages de l'architecture client-serveur. On voit cette tendance dans plusieurs systèmes de TA : Ariane-G5 avec CASH, la famille de Systran serveur avec un seul moteur installé accessible depuis plusieurs instances clients, *etc.*

En plus, on voudrait construire un méta-EDL universel qui peut se connecter et travailler avec plusieurs EDL à distance. Cette étape unifierait les données et traitements de divers EDL et fournirait un outil de convergence de conception et de réalisation de systèmes de TAO.

La première et la deuxième partie concernent les techniques d'implémentation d'un méta-EDL avec ces améliorations. La troisième partie décrit l'application de cette technique dans le moniteur EMEU_w.

4.1.1. Délocalisation par création d'un méta-EDL

Avant d'arriver à l'étude d'un méta-EDL universel, on voit d'abord des techniques de construction de méta-EDL dédiés à des EDL.

Pour construire un méta-EDL, il faut deux conditions : l'EDL central doit fournir une API pour qu'on puisse s'y connecter, et une instance d'un méta-EDL doit être installée sur la machine locale.

4.1.1.1. Technique externe de mise en réseau d'Ariane-G5 (réseau LIDIA)

Dans le système Ariane-G5, un linguiste peut tout traiter via le moniteur interactif, même à distance, avec un émulateur TN 3270. Cependant, le moniteur est de type à ligne de commande et édition de fichiers de paramètres. Cette limitation a été volontairement conservée en 1985, alors que des possibilités graphiques étaient devenues disponibles (SGDM avait été acheté et testé) pour que des aveugles ou amblyopes comme l'un des post-docs du GETA puissent continuer à utiliser Ariane-G5 via un Versabaille. On désire avoir une interface graphique, qui peut travailler sur les données sous des formats différents en local.

Pour ce faire, on a construit le réseau LIDIA (voir 3.2.2) pour donner l'accessibilité aux applications écrites en Ariane-G5 depuis l'extérieur. Cette couche est indépendante des programmes noyau, et elle a été utilisée dans différents projets : projet LIDIA pour la désambiguïsation interactive de TA par dialogue de désambiguïsation interactive sur des machines locales (voir 2.2.2.3), système CASH pour le développement à distance d'un système de TA pour Ariane-G5 (voir 4.1.1.2), système WICALE, un système de pilotage à distance de systèmes de TA (voir 4.1.2).

Le réseau LIDIA a transformé le système Ariane-G5 en un serveur et a permis la création de méta-EDL dédiés à Ariane-G5.

4.1.1.2. CASH (CASHrev) pour Ariane-G5

L'interface CASH (Commande d'Ariane Sous Hypertexte) est un méta-EDL du système Ariane-G5.

Cette interface a été écrite par E.Blanc [Blanc, 1999] au GETA. Elle permet de développer des linguiciels de TAO sous Ariane-G5 de façon déportée, asynchrone et enrichie par rapport à l'interface modale native.

CASH peut communiquer avec le système Ariane-G5 par échange de commandes dans la syntaxe d'Ariane-G5. De plus, comme il y a des dépendances entre des commandes (par exemple, la commande LISNOMTEXT pour lister les textes dans un corpus demande en paramètre le nom du corpus qui est fourni par la commande LISNOMCORP pour lister les corpus actuels), il nous faut stocker en local des copies des informations globales (nom des corpus, textes, couples de langues,...) sur les linguiciels (dictionnaires, grammaires, ...) présents dans la machine Ariane-G5.

La Figure 40 montre l'écran principal de CASH, qui reflète l'architecture linguicielle du système Ariane-G5 présentée en 3.1.2.3.

La disponibilité des linguiciels en local sous un format exploitable nous permet d'ajouter des fonctions de support de développement qui n'existent pas sur la machine d'Ariane-G5, comme : navigation dans les linguiciels (voir 4.1.2.2.2), indexage des dictionnaires, développement graphique des objets... Un linguiste peut développer des linguiciels d'une application de TA, exécuter la traduction, et déboguer les linguiciels.

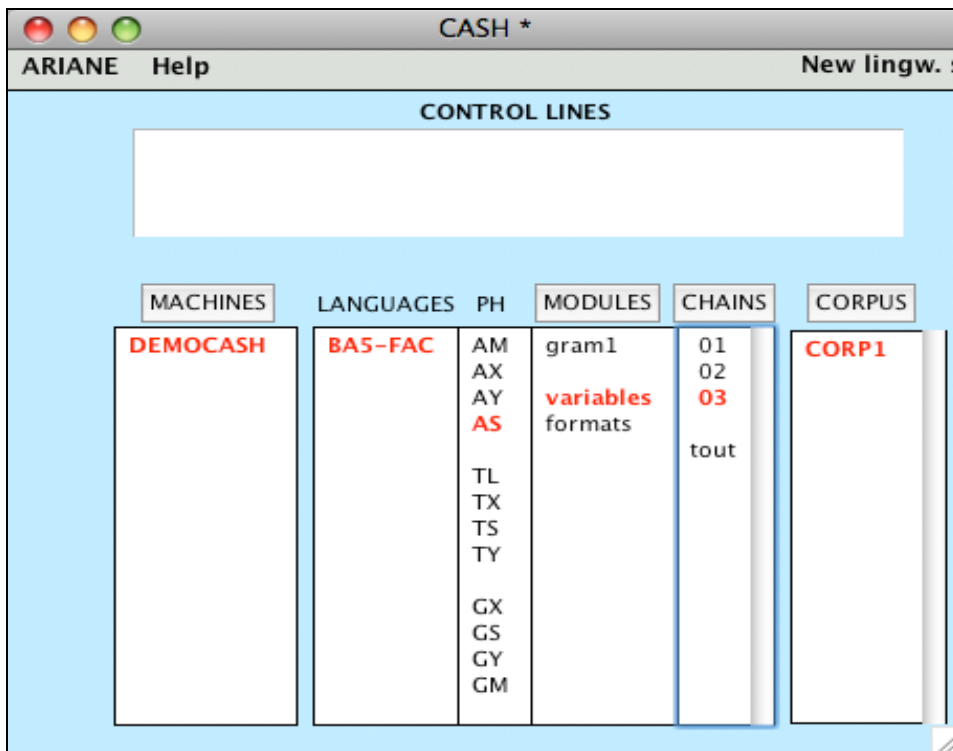


Figure 40: Interface de CASH

Quand l'utilisateur modifie ses linguiciels ou en crée de nouveaux, et les renvoie ensuite à Ariane-G5 pour test ou exécution, il peut également effectuer une mise à jour de ses copies en demandant à Ariane-G5 l'envoi de tel ou tel linguiciel.

Le format d'échange est basique et sous forme textuelle. Quand un utilisateur sélectionne une commande à envoyer via l'interface graphique, CASH produit une commande dans la syntaxe des commandes du moniteur d'Ariane et l'envoie à Ariane-G5. Le résultat reçu en retour est également un flot de texte dont il faut extraire la partie d'information utile avant de la ranger dans la copie des linguiciels en local.

Un exemple d'envoi d'une commande d'Ariane-G5 est présenté en annexe 2.1.

CASH a été implémenté en Hypercard, puis transformé en Revolution en 2008 (CASHrev).

4.1.2.TA hétérogène par méta-EDL générique (WICALE)

4.1.2.1.Métalangage pour définir les formats d'échange et d'interface

4.1.2.1.1.Motivations et objectifs

En 2003, bien que CASH soit pratique et fonctionne très bien, on voulait une interface plus portable (CASH avait été implémenté sous Hypercard, tournant uniquement sur Mac OS 9), et plus universel, c'est-à-dire pouvant qui peut se connecter à plusieurs systèmes de TA.

Le problème de portabilité a été résolu en CASH par le portage en Revolution qui tourne sous Windows, Linux et MacOS-X.

Le désir d'avoir une interface universelle restait toujours non réalisé. C'est pourquoi Damien Genthial a proposé de créer un méta-EDL générique permettant de piloter plusieurs EDL différents à distance. Cette idée a été réalisée sous la forme de WICALE (Web Interface for

Communication with All Lingware Environnement⁴⁰), dont la version 1.0 a été spécifiée et implémentée en Java dans le projet CNAM de Virginie Carpena en 2003-2004.

L'intérêt de WICALE 1.0 est d'offrir aux linguistes certains services de CASH, c'est-à-dire de leur permettre de travailler sur les données linguistiques en local, et de lancer les commandes des serveurs linguistiques.

WICALE peut piloter plusieurs EDL. Piloter un serveur linguistique, c'est :

- gérer sur le client une image des logiciels développés et mis en œuvre sur le serveur,
- synchroniser les données linguistiques),
- communiquer avec le serveur afin de récupérer le résultat d'une exécution sur le serveur.

WICALE représente une avancée vers l'unification entre plusieurs systèmes. En effet, quand on a des logiciels de systèmes différents présents dans une même machine, sous un format unique, on a une possibilité de convertir et d'unifier des logiciels d'un système à l'autre.

WICALE peut donc communiquer avec plusieurs systèmes. Pour cela, il offre un moyen générique pour modéliser les commandes à envoyer, analyser les résultats reçus, et décrire une interface associée à un système.

Cette généralité est permise par l'approche langage. On peut étendre WICALE à un nouvel EDL sans écrire de code Java, mais simplement en décrivant les commandes et les données de cet EDL. Pour ce faire, on utilise un métalangage exprimé en XML pour décrire les architectures de l'EDL connecté, ainsi que le format et la syntaxe de commandes d'échange, ainsi que la présentation de l'interface des composants logiciels. À partir de ces descriptions, WICALE génère les commandes selon la syntaxe associée à chaque système et les lui envoie selon les paramètres de connexion. Le résultat reçu est analysé et rangé dans les copies grâce à l'expression régulière écrite pour chaque commande.

On donne ci-dessous des exemples de ces descriptions dans deux cas d'usage, avec le système Ariane-G5 et le système PILAF. D'autres systèmes peuvent être ajoutés de la même façon.

4.1.2.1.2.Exemple pour Ariane-G5

4.1.2.1.2.1.Architecture de données

Comme on l'a vu, la hiérarchie d'un système écrit en Ariane-G5 (voir 3.1.2.3) est : MACHINE>DISQUE>LANGUE>PHASE>CHAÎNE D'EXÉCUTION>MODULE>...

On trouve une description correspondante :

<pre> <ARCHITECTURE> <STRUCTURE id="0"> <id>0</id> <niveau>1</niveau> <data>MACHINE</data> <libelle_data>LBL_MACHINE</libelle_data> <mise_a_jour>M</mise_a_jour><!--manuelle--> > <param_id></param_id> <suiv>1</suiv> <nom_fichier>Liste.xml</nom_fichier> <valeur_tous></valeur_tous> <valeur_def>CARPENA</valeur_def> <expression>.*</expression> </STRUCTURE> </pre>	<pre> <STRUCTURE id="1"> <id>1</id> <niveau>2</niveau> <data>DISQUE</data> <libelle_data>LBL_DISQUE</libelle_data> <mise_a_jour>M</mise_a_jour><!--manuelle--> > <param_id>0</param_id> <suiv>2;3;6</suiv> <nom_fichier>Liste.xml</nom_fichier> <valeur_tous></valeur_tous> <valeur_def>191</valeur_def> <expression>.*</expression> </STRUCTURE> ... </pre>
---	---

⁴⁰ Interface cliente générique pour le pilotage de serveurs linguistiques

<pre> <STRUCTURE id="2"> <id>2</id> <niveau>3</niveau> <data>LANGUE</data> <libelle_data>LBL_LANGUE</libelle_data> <mise_a_jour>A</mise_a_jour><!-- automatique commande LIENLANG--> <param_id>0;1</param_id> <suiv>3;4</suiv> <nom_fichier>Liste.xml</nom_fichier> <valeur_tous>*</valeur_tous> <valeur_def></valeur_def> <expression>*</expression> </STRUCTURE> </pre>	<p>id: Identificateur de composant niveau : Niveau de composant dans l'architecture générale data : Nom de composant libelle_data: Etiquette d'interface correspondante, c'est la clé dans fichier des messages. mise_a_jour: mode de mettre à jour, soit A (automatique par l'extraction de renvoi de serveur), soit M (manuel, à saisir) param_id : liste des composants précédents pour filtrer. ex. langues de telle machine, tel disque. suiv : liste des composants utilisant ce composant comme paramètre nom_fichier : nom de fichier d'image en local qui contient des résultat brut renvoyé par serveur à analyser valeur_tous : valeur dans tous les cas, si non utiliser, c'est vide. valeur_def : valeur par défaut expression : expression régulière à extraire l'information dans fichier image.</p>
---	---

Voici l'interface générée par WICALE 1.0 :

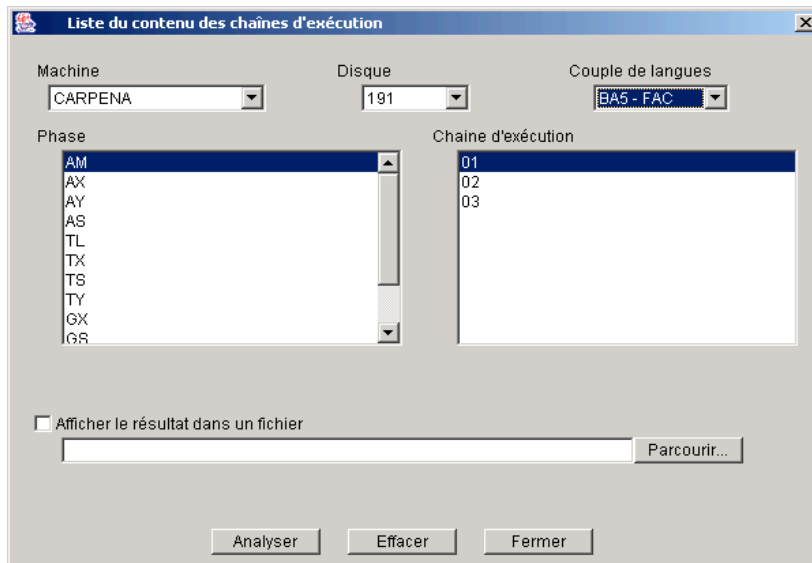


Figure 41: Interface générée par WICALE 1.0

4.1.2.1.2.2. Commandes

Comme CASH, WICALE communique avec Ariane-G5 par l'échange de commandes et de données. Avant de pouvoir générer les commandes avec la syntaxe d'Ariane-G5, on décrit ces commandes. Voici un exemple de description des deux commandes d'Ariane-G5 LISNOMCORP et LISNOMTEXT ; les autres commandes sont décrites de la même façon.

<p>LISNOMCORP: Liste des noms de corpus LISNOMCORP (<Langue source *, Liste des langues sources LIENLANG> , <Langue cible *, Liste des langues cibles LIENLANG> , <nom corpus *>).</p> <p>On voit ici que la commande LISNOMCORP demande</p>	<p>LISNOMTEXT: Liste des noms de textes pour un ou plusieurs corpus LISNOMTEXT (<Langue source *, Liste des langues sources LIENLANG> , <Langue cible *, Liste des langues cibles LIENLANG> , <nom corpus *, LISTE des corpus LISNOMCORP>).</p> <p>On voit la dépendance entre la commande LISNOMTEXT</p>
---	--

<p>en paramètre le couple de langues donné par la commande LIENLANG pour lister les noms de corpus existant pour tel couple de langues. Si l'on veut lister les corpus pour tous les couples de langues, on utilise (*) au lieu d'un code de langue précis.</p>	<p>avec LISNOMCORP et LIENLANG : LISNOMTEXT demande le nom de corpus que les textes sont associés (donné par LISNOMCORP) et/ou le couple de langue qui contient les textes à lister (donné par LIENLANG)</p>
<pre> <COMMANDE num_cde="1"> <num_cde>1</num_cde> <nom_cde>LISNOMCORP</nom_cde> <intitule_cde>Liste des noms de corpus</intitule_cde> <PARAMETRE_SAISIE> <PARAMETRE> <nom_param>Langues</nom_param> <libelle_param>LBL_LANG</libelle_param> <pos_lib_X>10</pos_lib_X> <pos_lib_Y>90</pos_lib_Y> <dim_lib_X>120</dim_lib_X> <dim_lib_Y>20</dim_lib_Y> <type_param>Popup</type_param> <valeur_def_param>*</valeur_def_param> <pos_X>30</pos_X><pos_Y>110</pos_Y> <dim_X>10</dim_X><dim_Y>80</dim_Y> <VALEUR_LISTE> <libelle_liste> LISTE_LANG </libelle_liste> <multiligne>>false</multiligne> <sep_multiligne></sep_multiligne> <valeur_liste>LIENLANG</valeur_liste> <pos_X>20</pos_X><pos_Y>60</pos_Y> <dim_X>160</dim_X><dim_Y>20</dim_Y> <initialise_liste>*-</initialise_liste> </VALEUR_LISTE> </PARAMETRE> </PARAMETRE_SAISIE> <SYNTAXE> <ETAPE> <mot_cle> TRAIT = LISNOMCORP (*) </mot_cle> <num_param></num_param> <expression></expression> <separateur>&retour_chariot;</separateur> <saisie_obligatoire> false </saisie_obligatoire> </ETAPE> </SYNTAXE> <RESULTAT> <resultat_OK>-> Tout est O.K.</resultat_OK> <resultat_type> -> Tout est O.K. </resultat_type> <ETAPE><nom_methode>find</nom_methode> <expr_deb> Liste des noms de corpus </expr_deb> <expr_corps> [A-Za-z0- 9.,?!()&caract_sub;&caract_accent; : = _ \$: +]* </expr_corps> <expr_fin>-LISTE TERMINEE-</expr_fin> <expr_concat>&retour_chariot;</expr_concat> <expr_replacement></expr_replacement> </ETAPE> </pre>	<pre> <COMMANDE num_cde="2"> <num_cde>2</num_cde> <nom_cde>LISNOMTEXT</nom_cde> <intitule_cde>Liste des noms de textes pour un ou plusieurs corpus</intitule_cde> <PARAMETRE_SAISIE> <PARAMETRE> <nom_param>Langues</nom_param> <libelle_param>LBL_LANG</libelle_param> <pos_lib_X>10</pos_lib_X> <pos_lib_Y>90</pos_lib_Y> <dim_lib_X>120</dim_lib_X> <dim_lib_Y>20</dim_lib_Y> <type_param>Popup</type_param> <valeur_def_param>*</valeur_def_param> <pos_X>30</pos_X><pos_Y>110</pos_Y> <dim_X>10</dim_X><dim_Y>80</dim_Y> <VALEUR_LISTE> <libelle_liste> LISTE_LANG </libelle_liste> <multiligne>>false</multiligne> <sep_multiligne></sep_multiligne> <valeur_liste>LIENLANG</valeur_liste> <pos_X>20</pos_X><pos_Y>60</pos_Y> <dim_X>160</dim_X><dim_Y>120</dim_Y> <initialise_liste>*-</initialise_liste> </VALEUR_LISTE> </PARAMETRE> <PARAMETRE> <nom_param>Corpus</nom_param> <libelle_param>LBL_CORPUS</libelle_param> <pos_lib_X>50</pos_lib_X> <pos_lib_Y>110</pos_lib_Y> <dim_lib_X>50</dim_lib_X> <dim_lib_Y>100</dim_lib_Y> <type_param>List</type_param> <valeur_def_param>*</valeur_def_param> <pos_X>50</pos_X><pos_Y>120</pos_Y> <dim_X>250</dim_X><dim_Y>150</dim_Y> <VALEUR_LISTE> <libelle_liste> LISTE_CORPUS </libelle_liste> <multiligne>>false</multiligne> <sep_multiligne></sep_multiligne> <valeur_liste>CORPUS</valeur_liste> <pos_X>20</pos_X><pos_Y>60</pos_Y> <dim_X>160</dim_X><dim_Y>20</dim_Y> <initialise_liste>0;1</initialise_liste> </VALEUR_LISTE> </PARAMETRE> </PARAMETRE_SAISIE> <SYNTAXE> <ETAPE> <mot_cle>TRAIT = LISNOMTEXT (</mot_cle> <num_param>Langues</num_param> <num_param>Corpus</num_param> <expression>.*</expression> <separateur>)&retour_chariot;</separateur> </pre>

<pre></RESULTAT> </COMMANDE></pre>	<pre><saisie_obligatoire>true</saisie_obligatoire> </ETAPE> </SYNTAXE> <RESULTAT> <resultat_OK>-> Tout est O.K.</resultat_OK> <resultat_type>-> Tout est O.K.</resultat_type> <ETAPE><nom_methode>find</nom_methode> <expr_deb>Liste des noms de textes</expr_deb> <expr_corps>[A-Za- z09&caract_sub;&caract_accnt;Punct]*</expr_corps> <expr_fin>-LISTE TERMINEE-</expr_fin> <expr_concat>&retour_chariot;</expr_concat> <expr_remplacement></expr_remplacement> </ETAPE> </RESULTAT> </COMMANDE></pre>
--	--

Partie syntaxe. Chaque commande peut contenir des paramètres décrits dans <PARAMETRE>. On peut décrire la présentation d'interface de ce paramètre par l'ensemble des éléments posX, posY, dimX, dimY pour les positions (X,Y) et les dimensions (largeur, hauteur) correspondantes. La valeur de ces paramètres peut être le résultat d'une autre commande, par exemple, LISNOMCORP pour le paramètre Corpus de LISNOMTEXT. On peut déclarer le type d'objet d'interface pour ce paramètre (liste, champs de texte, ...)

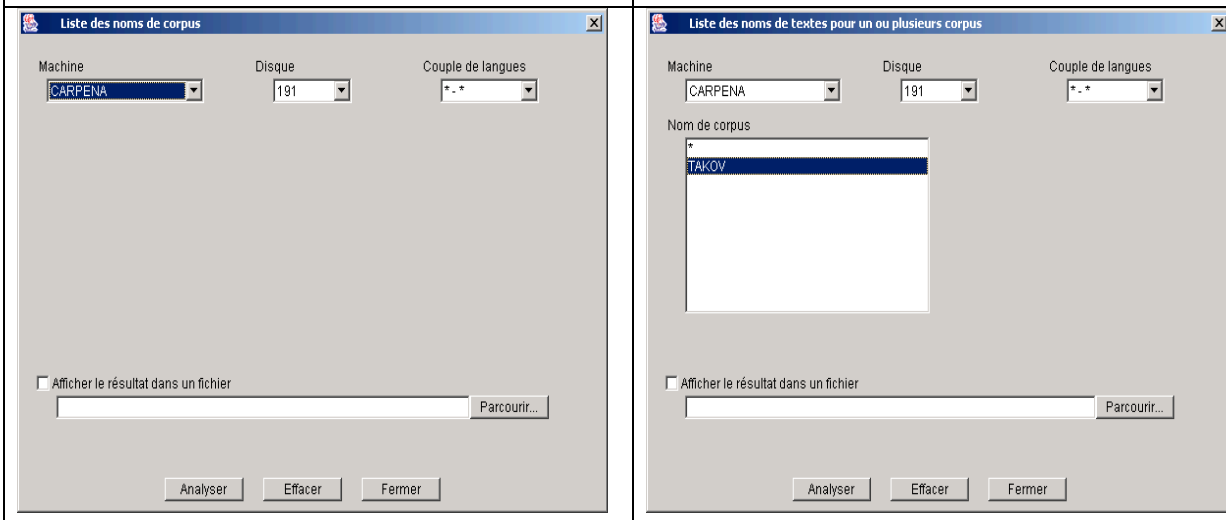
La présentation du libellé associé à un paramètre sur l'interface est décrite de la même façon. La valeur du libellé est le code d'un message dans un fichier de messages multilingues.

La syntaxe de la commande est décrite dans <SYNTAXE>. Par exemple, LISNOMTEXT prend deux paramètres donnés dans la description de paramètres *Langues* et *Corpus*, donc la syntaxe générée est : LISNOMTEXT LIENLANG LISNOMCORP

La commande générée est envoyée à Ariane-G5 selon le protocole défini dans la description de connexion du système. Le résultat reçu est considéré comme un flot de texte (voir annexe 2.1). On décrit dans la partie <RESULTAT> comment on peut analyser le texte brut et en extraire les informations pertinentes. On utilise ici une expression régulière pour enlever les noms de corpus et les noms de textes dans ces deux commandes. Le résultat extrait est stocké dans un fichier en local.

Voici l'interface générée à partir de la description de la commande LISNOMCORP

Voici l'interface générée à partir de la description de la commande LISNOMTEXT



4.1.2.1.3.Exemple pour PILAF

Le système PILAF (Procédures Interactives Linguistiques Appliquées au Français) créé par J. Courtin et X. Granjean dans l'équipe TRILAN en 1975-1976, est un logiciel de traitement

de textes écrits permettant l'analyse et la génération morphologiques, la lemmatisation, ainsi que la traduction au niveau morphosyntaxique (par exemple, vers du Braille ou vers une transcription phonétique). D. Genthial [Genthial, 1991] y a ajouté un analyseur de dépendances, et a porté la partie morphologie en C/C++. PILAF a été utilisé pour de nombreuses actions d'analyse de corpus. Son dictionnaire de 35 000 bases lui permet de reconnaître et d'engendrer environ 250 000 formes du français.

Ce programme est disponible sur Mac, Windows, Unix et sur le Web⁴¹. On ne peut actuellement utiliser que les fonctions d'analyse morphologique, de génération et de lemmatisation.

4.1.2.1.3.1. Architecture de données

Les données sont structurées dans PILAF de la manière suivante (voir Figure 42) : sur le serveur PILAF sont définis les dictionnaires, les catégories lexicales, les modèles, les règles et les variables pour le français.

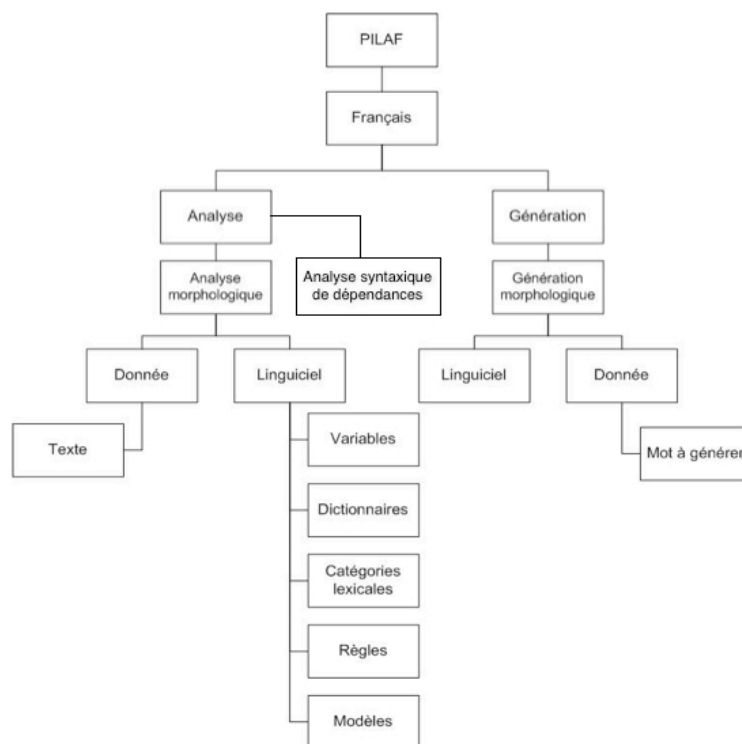


Figure 42: Architecture des composants linguiciels de PILAF

⁴¹ <http://www.daramechri.net/Pilaf/>

Table 3: Description de l'architecture des composants linguiciels de PILAF

<pre> <ARCHITECTURE> <STRUCTURE id="0"> <id>0</id> <niveau>1</niveau> <data>VARIABLE</data> <libelle_data>LBL_VARIABLE</libelle_data> <mise_a_jour>A</mise_a_jour><!--automatique--> <param_id></param_id> <suiv></suiv> <nom_fichier>Liste.xml</nom_fichier> </STRUCTURE> </pre>	<pre> <STRUCTURE id="1"> <id>1</id> <niveau>1</niveau> <data>CATEGORIE</data> <libelle_data>LBL_CATEGORIE</libelle_data> <mise_a_jour>A</mise_a_jour><!--automatique--> <param_id></param_id> <suiv></suiv> <nom_fichier>Liste.xml</nom_fichier> </STRUCTURE> ... </ARCHITECTURE> </pre>
---	--

4.1.2.1.3.2. Commandes

L'EDL PILAF est très particulier. Il n'existe pas de compilateur global, mais seulement une IHM de saisie et de modification interactive dans une ou deux phases prédéfinies pour le français. C'est un EDL exceptionnel car, pour l'instant, on ne peut pas échanger des ressources linguistiques de PILAF, mais seulement en saisir (dictionnaires, règles, variables morphologiques, catégories lexicales), données (texte à analyser, mot à générer). Le protocole de communication est http pour l'interface Web. L'ensemble des commandes de PILAF est relativement simple :

- **Visualiser la table des catégories lexicales** : Affiche la liste des catégories lexicales. Le paramètre table est égal à « catlex ».
- **Visualiser la table des variables morphologiques** : Affiche la liste des variables morphologiques. Le paramètre table est égal à « variables ».
- **Visualiser les composants** : Affiche la liste des composants (règles, dictionnaires, modèles). Dans la version Web de PILAF, on ne peut pas afficher ce type de données.
- **Analyse morphologique** : L'utilisateur saisit le jeu de caractères (ISO / ASCII), la lemmatisation (OUI / NON), le type de sortie (texte, HTML et LISP) et le texte ou le fichier à analyser. Le texte ou le fichier sont obligatoires. Par défaut, le jeu de caractères est « ISO », la lemmatisation « Non », et la sortie « HTML ».
- **Générer à partir d'un mot**. L'utilisateur saisit le jeu de caractères (ISO / ASCII), le mot, la catégorie et une ou plusieurs variables. Le mot est obligatoire. Par défaut, le jeu de caractères est « ISO ». La catégorie est sélectionnée parmi les données de la table des catégories lexicales. La ou les variables sont sélectionnées parmi les données de la table des variables morphologiques.

L'exemple suivant montre la description en WICALE d'une commande de visualisation des catégories lexicales.

Table 4: Exemple de commande pour visualiser les catégories lexicales

<pre> <COMMANDE num_cde="1"> <num_cde>1</num_cde> <nom_cde>tables.py</nom_cde> <intitule_cde>Liste des catégories</intitule_cde> <PARAMETRE_SAISIE/><!-- pas de parametres--> <SYNTAXE> <ETAPE> <mot_cle>tables.py?table=catlex</mot_cle> <num_param></num_param> <expression></expression> <separateur></separateur> <saisie_obligatoire>>false</saisie_obligatoire> </ETAPE> </SYNTAXE> </pre>

```

<RESULTAT>
<resultat_OK>&lt;PRE&gt;&lt;/resultat_OK>
<type_resultat>TEXTE</type_resultat>
<ETAPE>
<nom_methode>find</nom_methode>
<expr_deb>&lt;PRE&gt;&lt;/expr_deb>
<expr_corps>[0-9()&lt;&gt;}&lt;&lt;sub&gt;&caract_accent;]*&lt;/expr_corps>
<expr_fin>&lt;PRE&gt;&lt;/expr_fin>&lt;!--&lt;PRE&gt;-->
<expr_concat>&lt;/expr_concat>
<expr_remplacement>&lt;/expr_remplacement>
</ETAPE>
<ETAPE>
<nom_methode>replace</nom_methode>
<expr_deb>&lt;/expr_deb>
<expr_corps>&lt;B&gt;I&lt;B&gt;I&lt;B&gt;I&lt;PRE&gt;I&lt;PRE&gt;&lt;/expr_corps>
<expr_fin>&lt;/expr_fin>
<expr_concat>&lt;/expr_concat>
<expr_remplacement>&lt;/expr_remplacement>
</ETAPE>
</RESULTAT>
</COMMANDE>

```

À partir de cette description, WICALE interprète cette commande vers une URL :

http://serveur_PILAF/tables.py?table=catlex, où *serveur_PILAF* est remplacé par l'adresse du serveur PILAF (par exemple, www.daramechri.net/Pilaf). Cette requête est envoyée et une page HTML est renvoyée (Table 5). Les expressions régulières dans la partie <RESULTAT> extraient la liste des catégories lexicales concernées en deux étapes, extraction (*find*) et remplacement (*replace*) dans la description de la commande.

Table 5: Page HTML renvoyée par la commande de visualisation des catégories lexicales

<pre> <BODY BGCOLOR="WHITE"> <TABLE WIDTH="100%"><TR> <TD BGCOLOR="#666699" HEIGHT="30" VALIGN="CENTER"> Table des cat&eacute;gories lexicales </TD></TR></TABLE> <PRE> adv Adverbe subc substantif commun detp Déterminant-pronome det Déterminant subp Substantif propre adjq Adjectif qualificatif vide infi Infinitif ppt Participe présent ppas Participe passé verb Verbe xet Auxiliaire être xav Auxiliaire avoir pnt Point dpnt Deux points virg Virgule coco Conjonction de coordination prep Préposition locp Locution préposition </pre>	<pre> cocs Conjonction de subordination locs Locution conjonctive padv Pronom adverbial y en prl Pronom relatif sauf prlc Pronom relatif conjonctif ne Négation ne pas 2ème négation pas ce Pronom démonstratif ppe Pronom personnel phra Phrase vet Verbe être pnp Pronom non personnel loca Locution adverbiale ppf Pronom personnel fort adji Adjectif indéfini chf Chiffre date Date intj Interjection quan Quantification vav Verbe avoir de Préposition de à prep et à la (au) cls SuperCl </PRE> <TABLE WIDTH="100%"><TR><TD BGCOLOR="#666699">&nbsp;&lt;/TD></TR></TABLE> </BODY> </pre>
--	---

Après avoir appliqué l'extraction des informations de catégories et variables, le résultat est stocké dans un fichier en local (*Liste.xml*). Voici l'extrait de cette image en local :

<pre> <LST_LISTE> <LISTE nom_liste="VARIABLE"> <LISTE id=""> </pre>	<pre> <LISTE nom_liste="CATEGORIE"> <LISTE id=""> <VALEUR> adv - Adverbe</VALEUR> </pre>
---	--

<pre> <VALEUR> sin - Singulier</VALEUR> <VALEUR> plu - Pluriel</VALEUR> <VALEUR> fem - Féminin</VALEUR> ... <VALEUR> ref - Réfléchi</VALEUR> <VALEUR> cdn - Complément de nom</VALEUR> </LISTE> </LISTE> </pre>	<pre> <VALEUR> subc - substantif commun</VALEUR> <VALEUR> detp - Déterminant-pronom</VALEUR> ... <VALEUR> de - Préposition de</VALEUR> <VALEUR> à - prep et à la (au)</VALEUR> <VALEUR> cls - SuperCl</VALEUR> </LISTE> </LISTE> </LST_LISTE> </pre>
---	--

4.1.2.2.Exemple d'intégration de fonctionnalités utiles

WICALE 1.0 est un méta-EDL minimal pour la TA. Il offre aux linguistes les mêmes services d'échange de données que CASH, mais aucun autre. En ce sens, il est « minimal ». Par contre, V. Carpena l'a implémenté en Java avec une bonne paramétrisation, ce qui fait qu'il est portable sur plusieurs environnements et facile à étendre vers des architectures plus complexes.

Nous avons donc décidé de profiter de cet acquis et de conserver les services existants de WICALE 1.0 afin d'étudier et de construire de nouvelles fonctionnalités.

4.1.2.2.1.Édition « ouverte »

4.1.2.2.1.1.Techniques

Le cahier des charges de WICALE 1.0 ne prévoyait pas l'édition des images locales des logiciels source (en format texte et dans la syntaxe du LSPL concerné). Le but de WICALE 1.1 est de permettre à l'utilisateur d'éditer les composants logiciels source à l'aide d'un éditeur de son choix.

On s'est inspiré de ce qui est fait dans ARIANE-G5 (qui utilise XEDIT) :

- Par sécurité, l'utilisateur édite toujours une copie du composant en question.
- On a prévu deux modes V (Visualisation) et M (Modification) ; dans le premier, les modifications effectuées n'ont aucune conséquence.

4.1.2.2.1.2.Solutions

Pour implémenter la fonction d'édition, deux solutions sont possibles :

- Appel d'un autre éditeur : WICALE appelle un logiciel de traitement de texte disponible. L'utilisateur voit, édite et enregistre les données linguistiques source (non compilées) grâce à ce logiciel.
- Développement d'un éditeur : nous pouvons aussi développer nous-même un éditeur simple intégré à WICALE.

	Appel d'un autre éditeur	Développement d'un éditeur
Avantages	<ul style="list-style-type: none"> ▪Profite des fonctions de l'éditeur ▪Simple à réaliser 	<ul style="list-style-type: none"> ▪Indépendance à l'environnement
Inconvénients	<ul style="list-style-type: none"> ▪Dépendance à l'environnement 	<ul style="list-style-type: none"> ▪Coût de développement

Pour suivre notre « principe de délégation » et étant donné le peu de temps disponible, on a pris la solution suivante : appel à un éditeur narratif existant comme TextWrangler sur Mac ou Notepad sur Windows, ... Par expérience, des éditeurs plus sophistiqués risquent d'ajouter des informations non souhaitées au code source. Cela poserait problème.

Le plus grand problème de l'utilisation d'un logiciel extérieur à WICALE est qu'il faut connaître l'instant où l'édition est terminée. Une solution possible consiste à consulter le système d'exploitation pour contrôler la liste des tâches en cours, mais cette solution est forcément dépendante du système d'exploitation, donc de l'environnement. De plus, l'éditeur peut être toujours actif même si le fichier qui nous intéresse a été fermé.

Une autre solution consiste à enregistrer les informations concernant les fichiers ouverts dans une liste. WICALE ne contrôle alors pas directement le processus de l'édition par l'utilisateur. Cependant, quand WICALE intervient sur une copie locale, il vérifie la liste des fichiers ouverts et propose par précaution aux utilisateurs la mise à jour de ce fichier. Il faut évidemment faire attention à la possibilité qu'un utilisateur travaille directement sur un « fichier WICALE », et pas sur une copie de travail.

Le grand avantage de cette extension est l'indépendance par rapport vis à vis de l'environnement. De plus, on peut restaurer les copies en cours de modification suite à un arrêt (volontaire ou non).

4.1.2.2.1.3. Démonstration

À partir de l'écran de navigation, l'utilisateur clique sur le bouton d'édition pour modifier le contenu source du composant courant. Dans cet exemple, il s'agit de la déclaration des variables (DVS) dans la phase TL du couple de langue français-anglais BAC-FAC sur la machine CARPENA, disque 191 (voir Figure 43).



Figure 43: Ajout du bouton d'édition simple de WICALE 1.1

Quand l'utilisateur demande d'ouvrir le code source de ce composant, un message apparaît pour choisir le mode modification/affichage.



Figure 44: Message de demande du mode d'ouverture

S'il y a un éditeur défini dans les paramètres (dans cet exemple, c'est Notepad sous WINDOWS XP), il est appelé pour voir/modifier le composant. Si non, un éditeur par défaut est chargé.

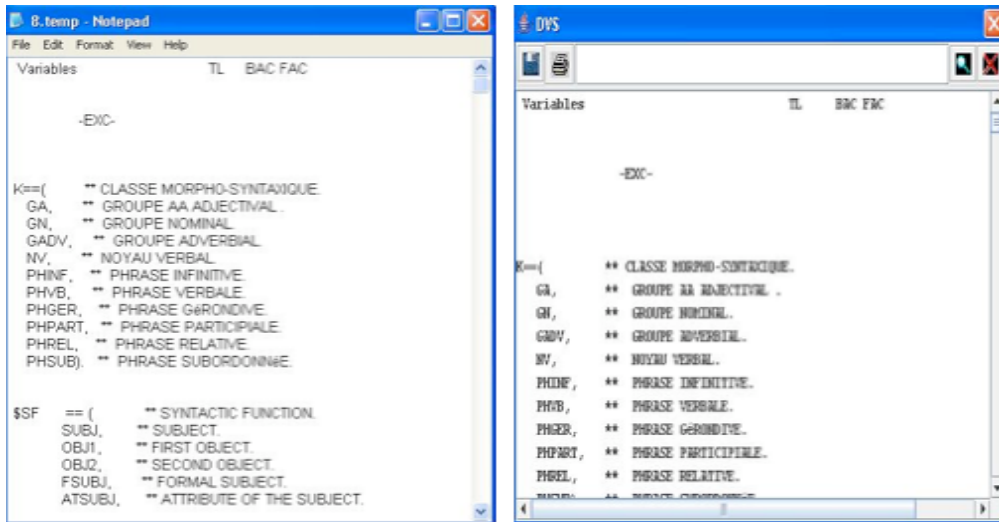


Figure 45: Exemple de l'éditeur simple Notepad et de l'éditeur par défaut

Sur la liste des fichiers ouverts, il existe un élément qui indique le fichier source avec le chemin vers le contenu et le nom du fichier temporaire correspondant.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- LIST FILE OPEN -->
<LST_EDIT>
+ <EDIT>
+ <EDIT>
+ <EDIT>
+ <EDIT>
+ <EDIT>
+ <EDIT>
+ <EDIT>
+ <EDIT>
+ <EDIT>
+ <EDIT>
- <EDIT>
<nom>8</nom>
<!-- Nom fichier temporaire 8.temp -->
<fichier>../DATA/ARIANE-G5/CARPENA191BAC - FACTL.xml</fichier>
<!-- Chemin vers copie locale -->
<path>../MODULE[@nom="DVS"]/contenu/</path>
<!-- Xpath vers source dans fichier -->
<taille>105</taille>
<!-- Taille du fichier -->
<moment>3920980989</moment>
<!-- Moment de modification transforme par Java -->
<id>1</id>
<!-- Identification de copie du source -->
</EDIT>
```

Ce travail a représenté 30h et environ 2 476 lignes de code en Java.

4.1.2.2.2. Navigation « à la Doxygen »

La fonction de navigation dans les linguiciels source est très pratique. Nous l'avons implémentée dans CASH ci-contre.

Dans l'exemple, on a cliqué sur la variable SUBA utilisée dans la définition de la procédure ADJ, d'où l'ouverture d'une autre fenêtre contenant la définition de cette variable.

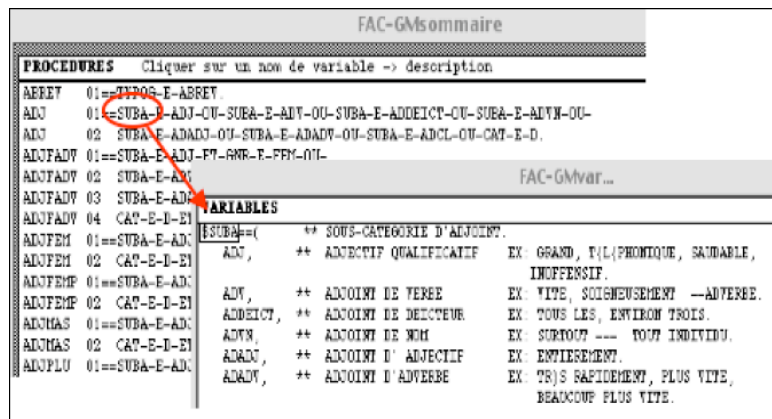


Figure 46: Navigation sous CASH

Il s'agit ici d'offrir une possibilité similaire à celle de CASH (implémentée par des scripts *ad hoc*). Nous avons proposé et implémenté une solution simple basée sur XML et inspirée de Doxygen⁴², un outil de génération de documentation.

Dans cette approche, un programme analyse et marque la liaison entre les occurrences et la définition de chaque élément. De plus, il prend en compte certains commentaires spéciaux (auteur, date, résumé, ...). Finalement, un générateur produit des fichiers HTML où toutes les occurrences dans le source deviennent des liens pointant vers la page contenant la définition. L'utilisateur navigue dans l'ensemble de ces fichiers HTML en utilisant n'importe quel navigateur Web.

Dans WICALE 2.0, la préparation des fichiers de navigation se passe de la même façon, en trois étapes:

- *transformation en XML du code source* des composants linguiciels, réalisée dans notre cas par le compilateur d'Ariane-Y ;
- *marquage* : parcours de la structure intermédiaire XML de chaque composant et insertion de liens entre définitions et occurrences ;
- *génération* d'un fichier html pour chaque composant, avec ajout à chaque occurrence d'un élément d'un lien vers la position de sa définition.

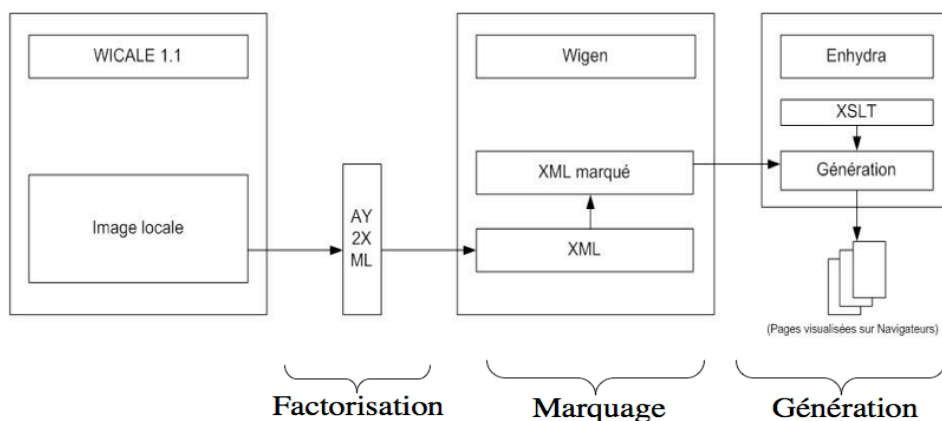


Figure 47: Trois étapes de la préparation des fichiers de navigation dans WICALE 2.0

Voici un exemple tiré d'un composant de "définition de variables" (définition d'un jeu de décorations linguistiques) :

<pre> ** Commentaires entre 2 étoiles et point. ** Transformation statique Jeu_1 --> Jeu_2. -DECVAR- dv . ** Nom du composant: DV. -DECO- deco . ** Nom du jeu : deco. MT ** Temps morphologique. ==(IMP, IPR, SPR, IPA, SPA, INF, PPR, PPA, FUT, CDL) . SEXE == (FEMININ, MASCULIN) . DGA == (SYN, ANA, NO) -CVAR- ** Transformation complémentaire (procédure) . CHGMT(C;@S) == -SI- MT(@S)-INC-IPR -ALORS- MT(C):=IPR; </pre>	<pre> -SNSI- MT(@S)-INC-SPR -ALORS- MT(C):=IPR; -SNSI- MT(@S)-INC-IPF -ALORS- MT(C):=IPA; -SNSI- MT(@S)-INC-SPF -ALORS- MT(C):=SPR; -SNSI- MT(@S)-INC-IPA -ALORS- MT(C):=IPA; -SNSI- MT(@S)-INC-FUT -ALORS- MT(C):=FUT; -SNSI- MT(@S)-INC-CDL -ALORS- MT(C):=SPA; -SNSI- MT(@S)-INC-IMP -ALORS- MT(C):=IMP; -SNSI- SUBV(@S)-E-INF -ALORS- MT(C):=INF; -SNSI- SUBV(@S)-E-PPR -ALORS- MT(C):=PPR; -SNSI- SUBV(@S)-E-PPA -ALORS- MT(C):=PPA; -FSI-. ... -FIN- </pre>
--	---

⁴² <http://doxygen.org>

Code	Description	Prca	Edtr	Link	Comment
CHGMT	LHGM1 == SI-ALORS-MI (C) = IPR ; SNSI-ALORS-MI (C) = IPA ; SNSI-ALORS-MI (C) = SPR ; SNSI-ALORS-MI (C) = IPA ; SNSI-ALORS-MI (C) = FUT ; SNSI-ALORS-MI (C) = SPA ; SNSI-ALORS-MI (C) = IMP ; SNSI-ALORS-MI (C) = INF ; SNSI-SURV (S)-E-IMP-ALORS-MI (C) = PPR ; SNSI-SURV (S)-E-PPR-ALORS-MI (C) = PPA ; SNSI-SURV (S)-E-PPA ; FSI				
CHGNR	LHGNR == SI-ALORS-GENUS (C) = FEM ; SNSI-ALORS-GENUS (C) = MAS ; SNSI ; FSI				
VOCAT	VOCAT == SI ; FSI				

Code	Description	Var	Edtr	Link	Comment
PLAD1	PLAD1 == (AVANT)				
RFLX	RFLX == (OBL, RECI, REC2, SEI, SE2)				
SUBA	SUBA == (ADJ, CARD, ORD)				
SUBJONC	SUBJONC == (VS)				
SUBN	SUBN == (NC, NP, RFP)				
SUBV	SUBV == (VF, INF, PPA, FPR)				
VOIX	VOIX == (ACT, PAS, EXPAS, IMP, IMPRXPAS)				
MT	MT == (IMP, IPR, SPR, IPA, SPA, INF, FPR, PPA, FUT, CDE)				
SEXE	SEXE == (FEMININ, MASCULIN)				
DGA	DGA == (SYN, ANA, NO)				
VNACT	VNACT == (NEU, MAS, FEM)				

Figure 48: Navigation sous WICALE 2.0

4.1.2.2.3. Extensions possibles

La plus grand principe de WICALE est la généricité. Cependant, dès qu'on veut améliorer WICALE, en y ajoutant une fonction comme l'édition des graphes, la visualisation ergonomique des objets (arbres, graphes), etc. Il faut spécifier pour chaque EDL associé.

Une solution est de développer un module générique pour de telles fonctions, ou de réaliser progressivement un LSPL pour cette fonction.

4.1.2.2.3.1. Édition « directe » d'objets linguistiques

Ce type d'éditeur vise à fournir aux linguistes un outil générique leur permettant de manipuler tout type de structures linguistiques à base de graphes : graphes (tels que les graphes UNL comme l'éditeur de Preedarat Jitkue (TER) dans SWIIVRE-UNL [Tsai, 2004]), arbres (tels que les arbres Ariane), automates (tels que les graphes de contrôle de ROBRA dans ARIANE-G5) à travers une interface à manipulation directe, avec intégration d'un langage de programmation spécialisé permettant la manipulation de ces structures par réécriture.

L'intérêt principal est que la représentation (graphique) est plus proche de la représentation mentale de l'utilisateur. Cela nécessite moins d'efforts cognitifs, car l'utilisateur dispose d'outils plus adéquats et intuitifs.

4.1.2.2.3.2. Reimplémentation progressive de LSPL dans WICALE

Dans le futur, on souhaite améliorer les LSPL d'Ariane-Y, et ceux qui auront été intégrés via WICALE, dans deux directions. L'une est de les simplifier pour permettre leur utilisation par des non-spécialistes de TALN [Bellynck, 2004], comme cela est possible avec TTEDIT de J.-C. Durand, dérivé de ROBRA, ou avec le LSPL SYLLA de syllabification de C. Del Vigna et V. Berment [Berment, 2004]. L'autre est de les étendre à la programmation classique, par exemple pour pouvoir écrire des compilateurs classiques avec des outils de TA.

L'idée est de produire à partir de chaque LSPL accessible via WICALE une famille de LSPL de divers degrés de « complexité conceptuelle » (ou un LSPL unique paramétrable par un niveau de complexité), et aussi de rendre sensibles les développeurs « naïfs » aux concepts sous-jacents aux (variantes des) LSPL utilisés.

Pour écrire les compilateurs de ces logiciels, on propose d'utiliser des générateurs de compilateurs comme javacc ou ANTLR.

4.1.3. Application au moniteur EMEU_w

4.1.3.1. Contexte et objectifs d'EMEU_w

Le résultat du projet EOLSS/UNL/FR (voir 2.3.2.2.2) présente deux parties. L'une est de rendre 25 articles d'EOLSS traduits et l'autre de mettre en place un service de déconversion UNL-français.

Comme la post-édition des prétraductions automatiques de ces 25 articles a été faite sous l'environnement SECTra_w, on aimerait avoir un portail unique qui permette à la fois de fournir l'accès à plusieurs composants disponibles : SECTra_w pour récupérer le résultat de traduction, PIVAX pour la partie lexicale, TRADOH++ pour le service de traduction automatique, etc. Nous avons donc décidé de créer un moniteur utilisable par le Web. C'est EMEU_w⁴³ (Environnement Multilingue pour EOLSS basé sur UNL) sur le Web. Ce travail a été réalisé avec l'aide du stage de TER de Master 1 informatique de Teban Zouhair en 2007-2008 [Zouhair, 2008]. Voici son architecture générale :

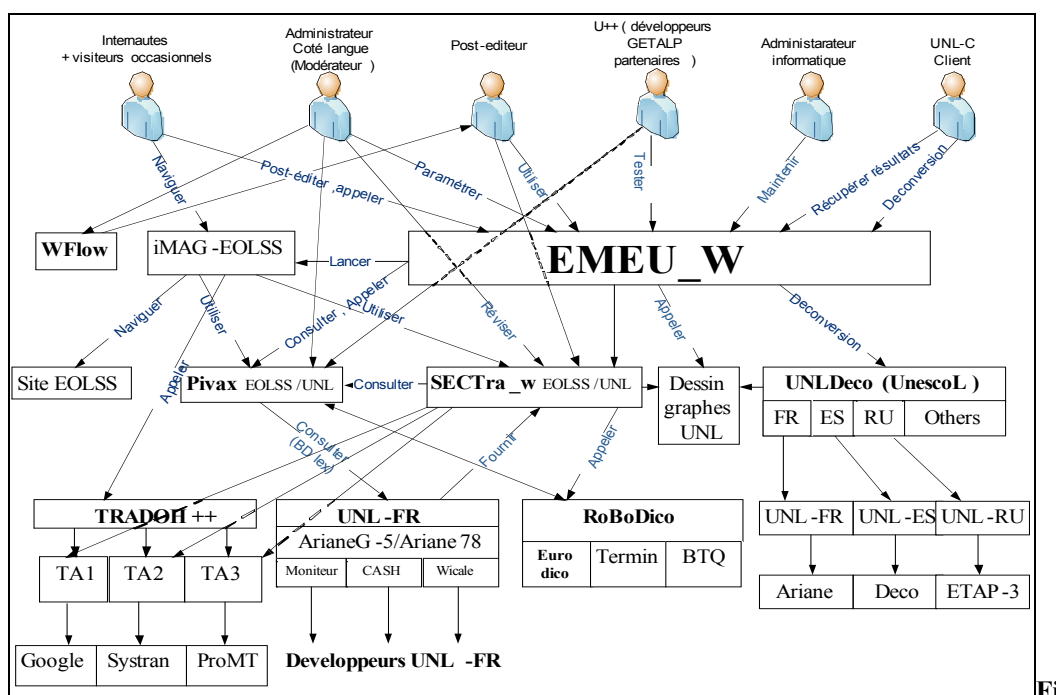


Figure 49: Architecture générale d'EMEU_w

Les classes d'utilisateurs sont :

UNL-C : ce sont les partenaires du projet EOLSS, fournisseurs des documents EOLSS. Ils peuvent réaliser la navigation sur les documents EOLSS via EMEU_w dans une langue, ils exploitent les résultats, et peuvent appeler la déconversion.

U++ (développeurs, GETALP et partenaires) : ce sont les personnes qui testent le bon fonctionnement du moniteur, et peuvent aider à la contribution lexicale via PIVAX.

Post-éditeurs : ce sont les personnes qui réalisent la post-édition soit en passant par le moniteur, ou directement sous SECTra_w. Elles peuvent réaliser la contribution lexicale via PIVAX.

Internaute : ils peuvent naviguer sur EOLSS via l'iMAG-EOLSS. Ils peuvent accéder à cette iMAG directement via EMEU_w, et contribuer (spontanément) à l'amélioration des traductions en passant dans le contexte de post-édition et d'amélioration des dictionnaires.

Administrateurs côté contenu (modérateurs) : ce sont les personnes qui distribuent les tâches aux post-éditeurs, révisent les travaux des autres utilisateurs, et récupèrent les traces des travaux pour chaque utilisateur.

Administrateurs informatiques : ils gèrent le moniteur EMEU_w au niveau informatique, peuvent créer des comptes aux nouveaux utilisateurs, s'occupent de la maintenance, et de la

⁴³ Le moniteur EMEU_w est accessible sur : <http://sway.imag.fr/unldeco/>

correction des différentes failles et bogues signalés par les autres utilisateurs de ce moniteur, et récupèrent les traces et mesures d'accès de tous les utilisateurs.

Tous les composants présentés dans le schéma ce-dessus ont déjà été présentés, à part Robodico. Il s'agit d'un module de recherche automatique des dictionnaires disponibles sur le Web. On le décrira en détail au 8.1.1.1.3.2.

Le fonctionnement d'EMEU_w ressemble à celui de WICALE. Une différence est que le moniteur EMEU_w est totalement sur le Web, et utilise donc des technologies plus spécifiques du Web.

4.1.3.2. Technique inspirée de WICALE

La technique d'implémentation d'EMEU_w aussi est similaire à celle de WICALE. On permet aux développeurs de décrire l'architecture, les commandes, la récupération des résultats des systèmes accédés sous forme d'un méta-langage à la XML.

Par rapport à WICALE, le protocole de communication dans EMEU_w est par défaut HTTP et le format d'échange entre systèmes est HTML.

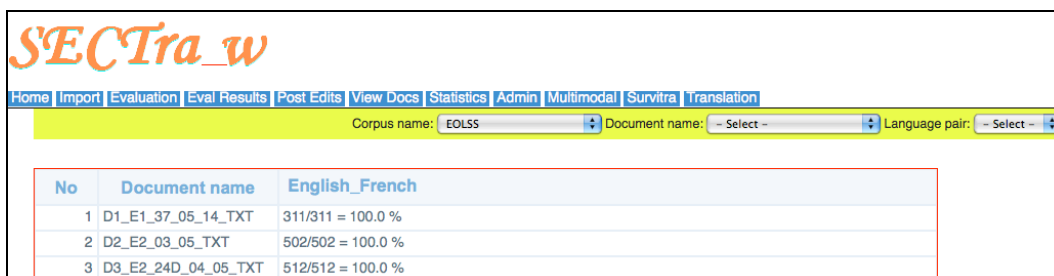
Au niveau de la récupération des résultats, on ajoute l'outil XPath pour analyser les pages HTML mieux que par des expressions régulières. Au niveau de la présentation, au lieu de décrire les positions des objets d'interface avec taille et position dans la fenêtre GUI de WICALE, on étend cela par l'ajout de transformations écrites en XSLT permettant aux utilisateurs de visualiser librement ce qu'ils veulent sur la page de résultat.

4.1.3.3. Validation et exemple de connexion avec SECTra_w

Pour une démonstration utilisant la connexion avec SECTra_w, on montre ici l'exemple de la commande de récupération de résultat de traduction des articles de EOLSS.

<pre><COMMANDE num_cde="4"> <num_cde>4</num_cde> <nom_cde>DOCUMENT_REV_LIST</nom_cde> <url>xwiki/bin/view/Corpus/PostEdit</url> <method>get</method> <delay>1000</delay> <num_cde_previous_list/> <PARAMETRE_SAISIE> <PARAMETRE> <nom_param>projName</nom_param> <libelle_param> Project </libelle_param> <dim_lib_parent> id_parent_a_attacher_dans_interface </dim_lib_parent> <dim_lib_lenght>20</dim_lib_lenght> <type_param>Hidden</type_param> <valeur_def_param> EOLSS </valeur_def_param> <local_valeur_list> /PROJECT_LIST/PROJECT </local_valeur_list> <dim_parent> id_parent_a_attacher_dans_interface </dim_parent> <dim_lenght>20</dim_lenght> </PARAMETRE> </PARAMETRE_SAISIE></pre>	<pre><RESULTAT> <item-group name="rev-projet"> <item name="no" type="xpath" out="text" expression="/html[1]/body[1]/div[1] /tbody[1]/tr/td[1]/div[1]" /> <item name="documentName" type="xpath" out="text" expression="/html[1]/body[1] table[1]/tbody[1]/tr/td[2]" /> <item name="documentStatus" type="xpath" out="text" expression="/html[1]/body[1]/ div[1]tbody[1]/tr/td[3]" /> </item-group> </RESULTAT> <PRESENTATION type="xslt"> <xslt-code> <!-- XSLT code ici ... --> </xslt-code> </PRESENTATION> </COMMANDE></pre>
--	---

Cette commande envoie une requête HTTP : <http://eolss.imag.fr/xwiki/bin/view/Corpus/PostEdit?projName=EOLSS> à SECTra_w. SECTra_w renvoie une page HTML (voir Figure 50)



The screenshot shows the SECTra_w web interface. At the top, there is a navigation menu with links: Home, Import, Evaluation, Eval Results, Post Edits, View Docs, Statistics, Admin, Multimodal, Survivra, Translation. Below the menu, there are three dropdown menus: Corpus name: EOLSS, Document name: - Select -, and Language pair: - Select -. The main content area contains a table with the following data:

No	Document name	English_French
1	D1_E1_37_05_14_TXT	311/311 = 100.0 %
2	D2_E2_03_05_TXT	502/502 = 100.0 %
3	D3_E2_24D_04_05_TXT	512/512 = 100.0 %

Figure 50: Page HTML renvoyé par SECTra_w

Les trois expressions Xpath nous rendent les trois informations concernées: numéro de document (article dans EOLSS), nom identifiant le document et statut actuel de la traduction. Une fois qu'on a extrait ces informations, on les met dans une structure interne et on appelle la feuille de transformation XSLT définie dans cette commande pour présenter le résultat dans l'interface d'EMEU_w.

Cela montre l'efficacité réelle dans le contexte de gestion de système. Si le système SECTra_w en cours de développement change sur la page de présentation, cela prend quelques minutes pour adapter le fichier XML du méta-langage. Même un non-informaticien peut le faire. Ensuite, le moniteur continue à rendre le service attendu.

Dans la partie d'en-tête de commande, on décrit le chemin URL, la méthode d'envoi et le temps d'attente. Si le système accédé ne répond pas dans le temps d'attente, on considère que le système est occupé et on renvoie la requête en peu plus tard. On a construit un mécanisme de gestion de commandes à envoyer dans une pile d'attente. On peut configurer la file d'attente avec une stratégie de file (FIFO), de pile (LIFO), ou toute autre stratégie programmable.

4.2. Réingénierie des LSPL en Ariane-Y

Le développement d'une application sur un système de TA demande certaines connaissances linguistiques et des compétences sur les LSPL associés, donc ce travail est réservé à des spécialistes, pas aux utilisateurs « naïfs », ni aux traducteurs professionnels provenant du monde de la THAM.

Pour pouvoir faire travailler des gens ayant des profils non-spécialistes en TALN, pour chaque LSPL il faut qu'on produise une classe de LSPL avec une diversité de degrés de « complexité conceptuelle » correspondante au profil de l'utilisateur : un utilisateur naïf, un traducteur, un informaticien, ...

La transformation des LSPL exige une réingénierie assez profonde des LSPL existants. Et, pour une utilisation ouverte dans le cadre d'un système de TAO hétérogène, à la fois pour la TA et la THAM, il faut que la réalisation de tels LSPL du système Ariane-Y prenne en compte des principes et contraintes appropriés.

4.2.1. Principes et contraintes du projet Ariane-Y

Pendant 10 ans, on a réfléchi aux principes de réalisation du projet sur plusieurs aspects. Ces idées ont évolué selon la disponibilité des ressources, l'avancement des outils et de la technologie.

On résume sur certains grands principes comme l'innovation logicielle pour la TA, l'organisation de la structure de travail et l'architecture du système.

4.2.1.1. Innovations logicielles pour la TA

4.2.1.1.1. Principe de permanence des objets en mémoire et modification incrémentale

Quand le système sera étendu pour traiter plusieurs applications de TA et de THAM, il devra être capable de traiter de grandes données dans des situations diverses comme la traduction en boucle de documents envoyés sans cesse par l'utilisateur de n'importe d'où. L'utilisation des traductions déjà exécutées ou post-éditées par un humain devient un facteur majeur.

Il faut donc que le système de TA fonctionne comme un serveur unique qui, une fois chargé, reste en attente. Tous les objets linguistiques sont également chargés en mémoire et l'interaction effectuée des modifications incrémentales sur ces objets.

Cela nous demande de définir une structure interne pivot qui contienne tous les objets possibles. Cette structure est également le format d'échange entre composants du système et entre le système avec modules et moniteurs externes.

4.2.1.1.2. Pas de limitation de taille

La taille d'une unité de traduction du système Ariane-G5 est de quelques pages standard à cause de la limite des moyens techniques à cette époque.

Quand le système Ariane-Y passe au contexte d'utilisation à la fois pour la TA et la THAM, on a de grands documents à traduire (200-400 pages comme cette thèse). Il faut que le système n'ait pas de limite de taille de traitement. Le « maximum » pratique peut être : une unité de traduction est 400 pages de 500 mots, soit 200 000 mots, ou 600 000 nœuds, au plus 50 niveaux de récursion, soit 20 000 000 nœuds, et 10 000 000 décorations différentes, donc la taille des nœuds est ~1Go, celle des décorations est 1Go, celle de la table de programmation dynamique est ~2,5Go.

Pour ce faire, la bibliothèque TabFich a été implémentée en C par J-C Durand pour résoudre ce problème. Il s'agit d'un mécanisme de gestion de mémoire utilisant les fichiers temporaires, et qui est similaire à MemFile de MS Windows, mais portable.

Un « tableau infini » est défini par un type d'enregistrement de taille fixe comme un type de page, un minimum et éventuellement un maximum de nombre de pages en mémoire, et un fichier. L'accès se fait via les numéros d'enregistrement, et la pagination comme la mise à jour du fichier sont réalisées de façon transparente par TabFich. On obtient « en prime » une persistance des objets.

4.2.1.1.3. Compilation en ANTLR et gestion des erreurs « sémantiques » au chargement

La méthode classique de construction d'un compilateur de LSPL est de construire un analyseur lexical (AL) et un analyseur syntaxique (AS). On charge la SI (Structure Interne) à partir des analyseurs. Cette approche est puissante mais l'implémentation doit être réalisée par un programmeur.

Une meilleure méthode consiste à utiliser un générateur de compilateurs comme ANTLR pour l'analyse lexicale et l'analyse syntaxique, et de traiter les erreurs sémantiques au moment du chargement de la structure interne (SI-API). Cette structure contient : (1) une représentation de la structure abstraite des objets (arbres, graphes, dictionnaires...), (2) informations utiles pour les algorithmes pour l'édition graphique, marques pour le parcours de graphes comme en systèmes-Q ou en ROBRA, *etc.*, et (3) des fonctions (API) de chargement, de tests sémantiques, d'export (sous différents formats dont un format XML « externe » et un arbre « interne » permettant un rechargement direct sans aucun test), et d'appel aux différents « moteurs » des LSPL, et éventuellement à des algorithmes directs (comme ceux utilisés en TA probabiliste).

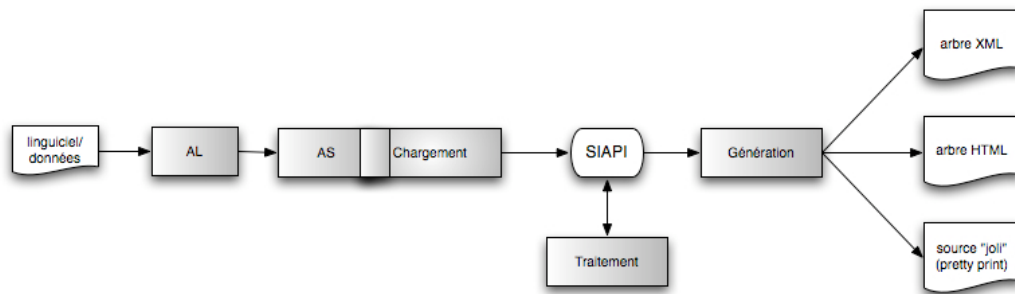


Figure 51: Construction d'un compilateur avec l'outil de génération des compilateurs

Parmi les outils disponibles, ANTLR⁴⁴ (ANother Tool for Language Recognition) est un des meilleurs outils grâce à ses outils de support (ANTLRWorks pour le développement de la grammaire syntaxique) et aux contributions de la communauté.

Un des problèmes de l'utilisation des générateurs de compilateurs comme ANTLR est le mélange de la grammaire avec la programmation, qui donne une grammaire difficile à lire ainsi qu'à maintenir. Pour éviter cela, on propose d'utiliser le mécanisme d'envoi de messages. Pour chaque règle de grammaire, au lieu d'ajouter directement du code source, on n'ajoute que des appels qui envoient des messages à la SI-API pour charger.

On voit ici l'exemple d'une règle de grammaire syntaxique avec des actions pour charger des objets dans la structure interne (SI):

<pre> primaireBooleen TIRET → TIRET NON → primaireBooleen PARG → POINT → expressionBooleenne → POINT → PARD listeParametree → relation → listeParametree </pre>	<pre> 363 primaireBooleen returns [int idx]: 364 (TIRET TIRET) { \$idx=addConditionTiretTiret(); } 365 (NON primBool=primaireBooleen) { 366 \$idx=addCond_NON_primaireBooleen(\$primBool.idx); } 367 (PARG POINT expBool=expressionBooleenne POINT PARD) { 368 \$idx=addCond_exp_primaireBooleen(\$expBool.idx); } 369 (lParamG=listeParametree rela=relation lParamD=listeParametree) { 370 \$idx=addCondRelationListeParams(\$lParamG.idx, \$rela.rel, \$lParamD.idx); } 371 ; </pre>
<p>Les appels à l'API (<i>addConditionTiretTiret</i>, <i>addCond_NON_primaireBooleen</i>, <i>addCond_exp_primaireBooleen</i>, <i>addCondRelationListeParams</i>) sont destinés à créer les objets équivalents dans la SI.</p>	

Un autre problème majeur est de traiter des erreurs. Avec ANTLR, on peut capturer les erreurs lexicales et syntaxiques renvoyées par la SI-API au cours du chargement. On peut aussi détecter des erreurs « sémantiques » sophistiquées. Par exemple, dans le dictionnaire de transfert français-UNL utilisé dans le déconvertisseur, on trouve à la fin de ce dictionnaire des entrées :

Numéro de la ligne	Contenu des entrées
39 386	<pre> ... [partir] {CAT (CATV), AUX (ETRE), VAL1 (GN) } </pre>

⁴⁴ <http://www.antlr.org>

39 387	"think(agt>human,icl>do)"; [penser]
39 388	{CAT(CATV),AUX(ETRE),VAL1(QU,GN)} "think(agt>human,icl>do,obj>entity)"; [penser]{CAT1(CATV),AUX(ETRE),VAL0(QU)} "think(agt>thing,icl>do,obj>uw)";

Les deux entrées « penser » ont été ajoutées en dupliquant l'entrée « partir », mais on a oublié de changer la valeur de l'auxiliaire de penser, qui est AVOIR, pas ÊTRE. De plus, la variable CAT1 n'existe pas dans la déclaration des variables.

La première erreur n'est pas détectable au niveau syntaxique, elle se situe au niveau du contenu linguistique. Par contre, la deuxième erreur, embêtant à détecter lors de l'analyse syntaxique, et est détectable au chargement, puisqu'à ce moment la déclaration des variables a été chargée dans la SIAPI.

4.2.1.2. Organisation de la structure de travail

4.2.1.2.1. SI (Structure Interne) = SA (Structure Abstraite) + EPA (Extensions pour Présentations & Algorithmes)

Afin de faire participer des non-spécialistes du TALN au développement des certains logiciels dans un système de TA, il faut que ces logiciels disposent de présentations adéquates et intuitives. Par exemple, on voit ici une règle de transformation d'arbre écrite en ROBRA (extrait de la thèse de S. Chappuy [Chappuy, 1993]):

<pre>MDEVEN: (*0,&NIV-1) (0(1(\$A,2,*),*(\$B,4),*,5(*V)) / 1: \$MODAL: 4: UL-E-"BE"; 5: SURV-INC-VEN; \$A; ADV; \$B; \$ADV == 0(5(\$A, 2, \$B, \$V)) / * <-- 1,3,4 / 2: 2, SF:=AUX; 5: 5, K:-VCL, VOICE:-PAS, UL:- "*VCL", NEG:-NEG(1) SUBV:=VB, NUM:=NUM(1) -ST- UL(2) -E- "COULD" -OU- UL(2) -E- "MIGHT" -ALORS- TENSE(5):-COND -U- PRES -STNON TENSE(5) := TENSE(2) -FST-</pre>	<p>Cette règle se compose d'un nom, d'une partie gauche (le schéma, qui est la condition) et d'une partie droite (l'image, qui est l'action).</p>
	<p>Le schéma se compose d'une description géométrique (schéma d'arbre) et d'une description algébrique (des conditions sur les décorations).</p>

La partie textuelle de la règle peut être éditée sous CASHrev, mais cela demande un entraînement cognitif important. CASHrev permet, en cliquant sur le texte des parties géométriques, de les obtenir sous forme graphique. Les formes graphiques correspondantes (transformation d'arbre, graphe de contrôle) sont en effet beaucoup plus intuitives et plus faciles à manipuler pour un utilisateur « naïf ».

L'étude de la présentation générique des objets linguistiques et de la séparation entre SI (Structure Interne) et SA (Structure Abstraite) a été commencée lors du M2R d'Émile Verdurand au GETA [Verdurand, 2005].

Le problème majeur de la représentation et de l'édition graphique des objets linguistiques est la complexité de leurs structures : arborescences étiquetées (arbre d'Ariane), hypergraphes (UNL), treillis (amphigrammes), dépendances entre des éléments (dictionnaires, corpus, formats contenant des variables qui sont définies ailleurs), *etc.*

Du fait de cette diversité⁴⁵, de nombreux outils (éditeurs,...) spécialisés pour certaines structures ont été développés, mais ils deviennent obsolètes lorsqu'on change de domaine.

Les recherches actuelles (ou passées) sur la généricité n'ont pas apporté de réponses réellement satisfaisantes, la généricité étant trop générale ou trop restreinte.

Après avoir considéré plusieurs approches, nous sommes arrivé à une idée de base simple et puissante : utiliser pour chaque type d'objet comme structure interne (SI) non pas la structure abstraite (SA) de la forme textuelle habituelle, mais une structure abstraite décrivant directement son organisation interne, en l'enrichissant de sous-structures décrivant les paramètres essentiels de la présentation dans la ou les vues de l'objet en question.

Cette solution permet aussi une généricité à la sortie. On peut définir comme on veut la présentation par des vues différentes. De plus, à partir de la SI, on peut intégrer des informations spécifiques (EPA) pour l'éditeur ou utilisées par les algorithmes travaillant sur cette SI. Par exemple, dans les arbres de ROBRA, on ajoute sur chaque nœud des marques pour les règles appliquées, et un entier pour le niveau de récursion.

On illustrera cela en présentant la SI des systèmes-Q reimplémentés au 4.2.2.

4.2.1.2.2. Rôle de XML

Dans la situation où le système de TA est utilisé par des utilisateurs non-spécialistes, on voudrait que les linguiciels soient disponibles en format exploitable et navigable (comme HTML) sous un outil populaire comme un navigateur de Web standard.

Dans l'architecture décrite, un système de TA joue un rôle d'agent et les données et commandes échangées entre des systèmes sont parcourues comme un flot. Il est donc indiqué de les mettre dans un format bien supporté par la technologie Web.

Toutes ces motivations nous conduisent à réaliser un export des linguiciels vers XML pour la navigation, la sérialisation et l'échange.

J-C. Durand a implémenté un tel module (AY2XML). Nous avons l'utilisé dans WICALE 2.0 pour ajouter la fonction de navigation dans les linguiciels en local (voir 4.1.2.2.2).

4.2.1.2.3. REXX et REXXstack

REXX (REstructured eXtended eXecutor) est un langage de programmation de haut niveau développé par IBM [Cowlshaw, 2009] depuis 1979. REXX ne contient qu'environ 23 mots clés réservés (comme call, parse et select) avec ponctuations et formats minimaux, et un seul type de données, la chaîne. Le point fort de REXX est qu'il est simple à lire, à programmer et à tracer/déboguer.

⁴⁵ Ce texte a été extrait du rapport M2R UJF de É. Verdurand.

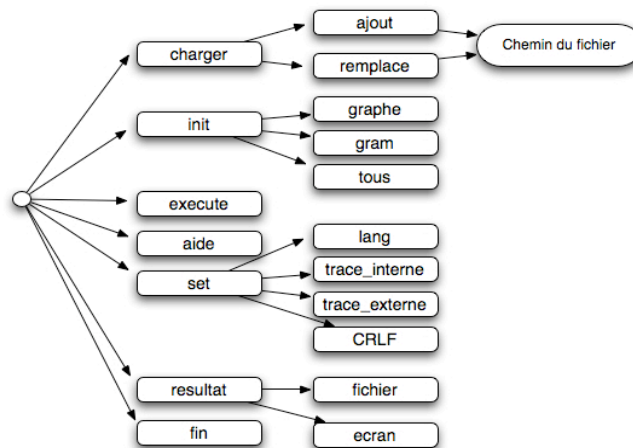
Les autres choses qui font la puissance de REXX sont : les variables pointées (a.b.c.d, a.b ...), les structures de contrôle, les instructions puissantes de « passage » de chaînes, le lien avec le langage C.

Voici un exemple de REXX pour créer des commandes par leurs paramètres associés. Ce code doit être fait en 50 lignes de code équivalent en C ou Java. En plus, la programmation d'un tel programme ressemble à une description logique alors que la programmation traditionnelle demande certains efforts cognitifs.

```

/*Liste des commandes et de leurs arguments du moniteur des systèmes-Q */
/*Liste des commandes (dans l'ordre alphabétique par commodité) */
comFr = 'charger init status execute aide set resultat fin'
/*Le premier argument 'constant' des commandes à 1 ou 2 arguments: */
comFr.charger = 'ajoute remplace test'
comFr.init = 'tous graphe gram'
comFr.status = 'tous graphe gram'
comFr.execute = ''
comFr.aide = ''
comFr.set = 'lang trace_interne trace_externe CRLF'
comFr.resultat = 'fichier ecran'
comFr.fin = ''
    /* deuxième argument: */
comFr.charger.ajoute = ''
comFr.charger.remplace = ''
comFr.charger.test = ''
    
```

La carte syntaxique équivalente :



Voici un autre exemple pour montrer la puissance de « passage » de REXX :

phrase = « I think, therefore I am [I think]. »	
<p>Programmation traditionnelle en Java</p> <pre> String remaining-words = phrase; int i=0; while (remaining-words <> "") { remaining-words:=strip(remaining- words,"Leading"); end-pos:=pos(" ",remaining-words) ; word[i]:=subString(remaining-words, 1,end-pos); i++; remaining-words:=subString(remaining- words,end-pos); } </pre>	<p>en REXX</p> <pre> do i=1 for words(phrase) word.i = word(phrase,i) end </pre>
On peut diviser la phrase en des parties par une seule instruction:	

```

parse var phrase precondition ',' consequence '[' qualiflier ']'
Le résultat:
precondition: "I think"
consequence: "therefore I am"
qualiflier: "I think"
    
```

On peut développer en REXX sous l'éditeur THE⁴⁶ (The Hessling Editor, un éditeur compatible avec XEdit implémenté par Hessling), et on peut appeler des bibliothèques dynamiques depuis REXX.

Ce langage a été conçu pour pouvoir être utilisé comme le langage de macro ou de script de n'importe quel système. Il en existe plusieurs variantes pour plusieurs plates-formes (Linux, Windows, Mac,...) et on a développé également des extensions de REXX vers des langages de programmation par objets (Object REXX).

De plus, REXX fournit un mécanisme simple mais efficace de communication entre applications ou machines par échange de listes de chaînes de caractères (commandes, données). À la réception, on peut gérer les commandes par les piles-files (REXXstacks) implémentées dans REXX.

On voit ici un exemple d'utilisation d'une pile-file externe de REXX comme une file LIFO par deux programmes lancés indépendamment. On peut faire communiquer deux programmes, deux machines avec des instructions simples, sans avoir besoin de connaissances techniques sur le protocole d'échange entre les machines, sans tampons (buffers), et sans bibliothèque spéciale (Writer, WriterStream, Reader, ReaderStream, ...) au contraire de la programmation traditionnelle.

ErireQ.rexx	LireQ.rexx
<p>Fonction : Créer une queue, mettre les noms de fichiers donnés par la commande 'ls', quitter.</p> <p>Ces éléments sont relus dans un autre processus (LireQ.rexx).</p>	<p>Fonction : Intervenir dans la queue créée par ErireQ.rexx sur une même machine, imprimer ses éléments.</p>
<pre> 'rxstack&' /* une seule instance est créée même si on réexécute */ trace o /* Ne pas demander la trace */ /*création avec options par défaut (localhost (127.0.0.1) et port (5757)): */ cq=rxqueue('Create','maFileExt@') /* 'say' pour afficher à l'écran */ say " nom de la file externe: " cq rc = rxqueue('Get') /* lister les fichiers dans le répertoire*/ 'ls -l rxqueue' cq '/lifo' nb = queued() /* nombre d'éléments de la pile mis sur la pile: */ push nb exit </pre>	<pre> trace o rc = rxqueue('Set', 'maFileExt@') say "nom de la file remplacée:" rc rc = rxqueue('Get') say " nom de la file courante: " rc if(queued() <> 0) then do parse pull nb /*nombre d'éléments en début de la pile-file */ do i = 1 to nb parse pull line say ' fichier' i ':' line end i end else say ' la pile est vide.' exit </pre>
<p>nom de la file externe: MAFILEEXT@127.0.0.1:5757</p>	<p>nom de la file courante: MAFILEEXT@127.0.0.1:5757 fichier 1 : test.the fichier 2 : ps.out fichier 3 : itf.txt ...</p>

⁴⁶ <http://hessling-editor.sourceforge.net/>

Le choix de REXX pour implémenter de grandes parties d'Ariane-G5 était tout à fait justifié, non seulement car on travaillait sur des machines IBM, mais encore par sa facilité d'utilisation, de maintenance, et de portabilité.

Le projet Ariane-Y continue à profiter de cet outil. On essaie également de faire migrer d'autres modules implémentés dans des langages proches comme PL/1, EXEC, EXEC 2 vers REXX pour qu'on ait un système plus homogène au niveau du langage de programmation.

Le mécanisme de REXXstack est compatible avec l'extension d'architecture vers des serveurs/services. On évite de suivre la maintenance des composants de fournisseurs des services de communication de réseau LIDIA, et on le remplacerait par REXXstack avec un coût d'implémentation relativement moins chère et l'on obtiendrait un système complet et homogène.

4.2.1.3.Principe d'architecture

4.2.1.3.1.Délégation

Avec le désir d'intégration des fonctions de THAM dans le système de TA (voir le chapitre 2), on voudrait utiliser des composants en vue des systèmes autonomes. C'était bien le cas des LSPL d'ATLAS, LT ou l'éditeur TTEDIT, qui ont été conçus pour une tâche spécifique, mais qui ont été intégrés au système Ariane-G5 comme des composants auxiliaires, et on les intervient par échange des données via un format connu.

On suit ce principe dans le système Ariane-Y. On délègue certaines fonctions provenant de THAM par les composants associés comme : développement de connaissance lexicale par PIVAX, post-édition, révision, gestion d'une MT par SECTra_w. Un autre problème est la synchronisation de données entre ces systèmes.

4.2.1.3.2.Agents à gros grains

Pour utiliser d'autres systèmes, il nous faut une architecture logicielle distribuée adéquate.

L'architecture est le plus moderne qu'on vise à construire est une architecture orientée par le service (voir 3.4.1.3) où chaque système fonctionne indépendamment sur chaque site et propose certaines données (plusieurs instances de PIVAX, chaque instance gère un ensemble des dictionnaires spécialisés). Les systèmes participent à un flot de données et la communication et la synchronisation sont faites par un agent implémenté sur chaque site.

4.2.1.3.3.Possibilité de plusieurs moniteurs en parallèle

L'idée est qu'on permet à la construction des moniteurs dédiés pour des tâches spécifiques que l'utilisateur le souhaite de les avoir. D'ailleurs, au niveau du système, on permet également d'intégrer certaines fonctionnes d'Ariane par la programmabilité.

Avec l'architecture utilisant le portail unique de gestion des commandes via REXXstack dans Ariane-Y, on résout parfaitement ce problème.

4.2.2.Implémentation des LSPL : exemple des systèmes-Q

Pour illustrer les techniques d'implémentation de LSPL mentionnées dans la partie précédente, il nous faut les réaliser pour un LSPL réel.

On prend les systèmes-Q d'A. Colmerauer [Colmerauer, 1971] pour la nécessité et puissance pratique qu'ils apportent dans des traitements de TALN.

Les systèmes-Q sont simples et puissants. On avait une version installée au GETA en Fortran et expérimentée par plusieurs applications. Cependant, à cause des raisons de la diversité des installations, on a perdu les sources complets des systèmes-Q.

De plus, on veut que les systèmes-Q soient réimplémentés par les techniques modernes permettant la portabilité, l'utilisation des caractères Unicode, l'absence de limitation à la taille de la mémoire de traitement, ainsi que des améliorations.

On rappelle d'abord la définition des systèmes-Q, puis on présente leur implémentation détaillée avant de donner quelques exemples fonctionnels.

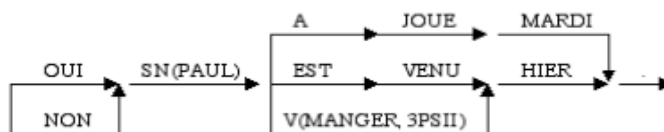
4.2.2.1.Systèmes-Q

Les systèmes-Q ont été proposés par Alain Colmerauer à l'université de Montréal. L'application la plus connue des systèmes-Q est le système de TA MÉTÉO [Chandioux et Guéraud, 1981] de traduction des 11 000 mots par jours de bulletins de météo au Canada (1977-1985) et TAUM-AVIATION [Isabelle et Bourbeau, 1984] de traduction des manuels de maintenance d'avion (1976-1980).

4.2.2.1.1.Rappel de la syntaxe et la sémantique opérationnelle des systèmes-Q

4.2.2.1.1.1.Syntaxe

Les systèmes-Q sont un ensemble de règles permettant de faire certaines transformations sur des graphes orientés. On appelle ces graphes « graphes-Q ». L'ensemble de tous les graphes est nommé « données-Q », et l'ensemble des règles s'appelle « traitements-Q ». Voici un exemple de graphe-Q de présentation des chaînes:



Le graphe-Q ci-dessus présente l'ensemble des chaînes :

- OUI + SN(PAUL) + EST + VENU + HIER + .
- OUI + SN(PAUL) + A + JOUE + MARDI + .
- OUI + SN(PAUL) + V(MANGER, 3PSII) + HIER + .
- NON + SN(PAUL) + EST + VENU + HIER + .
- NON + SN(PAUL) + A + JOUE + MARDI + .
- NON + SN(PAUL) + V(MANGER, 3PSII) + HIER + .

Le graphe-Q ne contient qu'une seule entrée et une seule sortie et il n'y a pas de boucle dans graphe. Sur chaque arc, on attache un objet représentant soit une étiquette, soit un arbre. Dans l'exemple précédent, 'OUI', 'NON', 'A', 'JOUE', ... sont des étiquettes. SN(PAUL) est un arbre avec un seul fils d'étiquette 'PAUL'. V(MANGER, 3SPII) est un arbre et ses fils sont une liste des étiquettes 'MANGER' et '3SPII'.

Pour présenter formellement un graphe-Q, on annote les nœuds par des numéros uniques. On obtient la suite des arcs ci-dessous (en colonne de gauche). Afin d'abrégier l'écriture, on permet de remplacer toute sous-suite de nœuds sans carrefour par l'étiquette (+), on a une présentation du graphe-Q en question dans la colonne de droite :

-1- OUI -2-	-1- OUI -2-
-1- NON -2-	-1- NON -2-
-2- SN(PAUL) -3-	-2- SN(PAUL) -3-
-3- EST -4-	-3- EST + VENU -4-
-4- VENU -5-	-3- V(MANGER, 3PSII) -4-
-3- V(MANGER, 3PSII) -5-	-4- HIER -5-
-5- HIER -6-	-3- A + JOUE + MARDI -5-
-3- A -7-	-5- . -6-
-7- JOUE -8-	
-8- MARDI -6-	
-6- . -9-	

Les traitements-Q sont des règles de transformation sur les données-Q dont la syntaxe est: $\text{partie-gauche} == \text{partie-droite}/\text{conditions}$.

La partie gauche décrit l'ensemble des chaînes du graphe sur lesquels cette règle peut être effectuée. On vérifie toutes les règles, l'une après l'autre sur données-Q. Si on trouve un chemin qui correspond à la partie gauche d'une règle, on ajoute le chemin fabriqué par la partie droite entre l'origine et l'extrémité du chemin instanciant la partie gauche de ce chemin. La condition contient des contraintes pour effectuer cette règle par la validation des expressions booléennes. Une expression présente des opérants (arbre, liste, chaîne) et des opérations (-DANS-, -HORS-, =, ≠, -ET-, -OU-, -NON-, ...)

L'exemple suivant démontre un transfert de traduction basique français-anglais :

```
OUI == YES.
NON == NO.
EST + VENU + HIER == V(COME, PAST) + YESTERDAY.
EST + VENU + HIER == V(COME, PRESENT_PERFECT, SIN) + YESTERDAY.
V(MANGER, 3PSII) = V(EAT, 3PSII).
...
```

A. Colmerauer a introduit la variable par la notion d'objet paramétré. Un objet paramétré est un arbre (ou une liste) paramétré, qui représente l'ensemble de tous les arbres (ou listes) que l'on peut obtenir, en substituant à chacun de ses « paramètres étiquettes » une étiquette quelconque. À chacun de ces « paramètres arbres », on a un arbre quelconque et à chacun de ses « paramètres listes », on a une liste quelconque. Par « substituer », on entend que l'on remplace différentes occurrences d'un même paramètre par la même chose. Par exemple :

```
V(I*, 3PSII) == V(EAT, 3PSII) / I* -DANS- (MANGER, BUFFER).
V(J*, 3PSII) == J* + S / J* -HORS- (GO, DO).
--           == J* + ES / J* -DANS- (GO, DO).
```

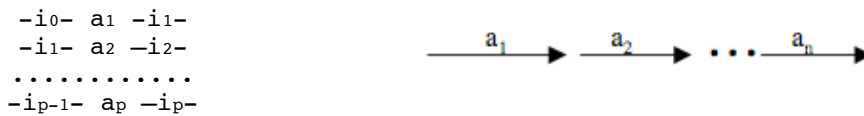
Les systèmes-Q sont transversales et modulables. On peut séparer le traitement en une suite de phases où le graphe d'entrée d'une phase est le résultat d'une autre.

Le détail de syntaxe se trouve dans [Colmerauer, 1971].

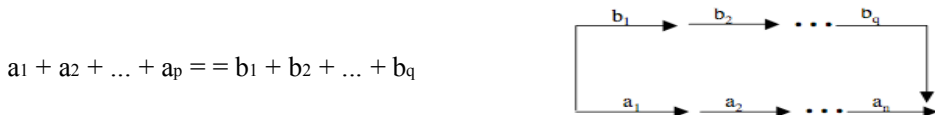
4.2.2.1.1.2.Sémantique opérationnelle

Le processus de traitement est une transformation d'un graphe de chaînes en un autre par des règles via deux étapes : addition des flèches et suppression des flèches inutiles.

On considère toute suite de flèches de la forme suivante dans un graphe-Q :



Si on a une règle sans paramètres dans traitements-Q simplifiés:



Dans la phase d'addition, on cherche à appliquer les règles sur tous les chemins du graphe-Q, puis on marque les chemins utilisés.

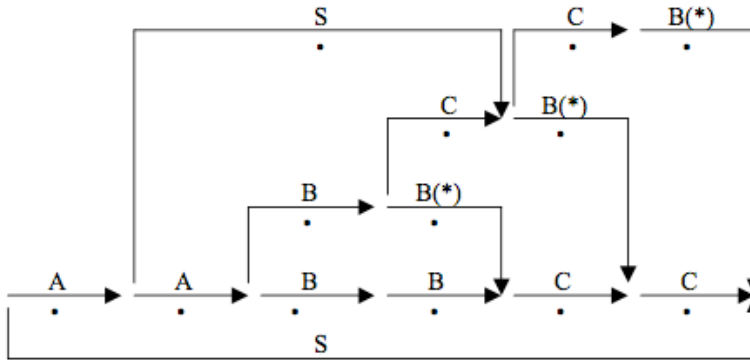
Dans la phase de suppression, on supprime les chemins marqués utilisés lors de la phase d'addition et les chemins qui ne sont plus inclus dans un chemin allant de l'entrée à la sortie du graphe.

Voici un exemple complet sans variable extrait de [Colmerauer, 1971] dans lequel on a un traitement-Q avec des règles sur le graphe de chaîne à droite :

$A + B + C = S.$
 $A + S + B(*) + C = S.$
 $B(*) + C = C + B(*).$
 $B + B = B + B(*).$



Après la première phase, on obtient le graphe :



La deuxième phase enlève toutes les flèches annotées par le point, on a le résultat de transformation de :

-0- A + A + B + B + C + C -1-
en

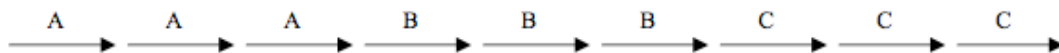
-0- S -1-

Dans les systèmes-Q complets qui ont des paramètres, conditions, ... il faut instancier ces paramètres au cours de réalisation et évaluer la condition avant d'ajouter le nouveau chemin.

On voit ici un exemple de système-Q avec des variables :

Graphe-Q en entrée :

-0- A + A + A + B + B + B + C + C + C -1-



Traitement-Q :

** EXEMPLE DE SYSTEME-Q.

$A^* + A^*(U^*) == A^*(1, U^*) / A^*$ -DANS- A, B, C.

$A(U^*) + B(U^*) + C(U^*) == S(1, U^*)$.

Il est équivalent au système-Q simplifié qui contient notamment les règles:

$A + A == A(1).$

$A + A(1) == A(1).$

$A + A(1,1) == A(1,1).$

.....

$B + B == B(1).$

$B + B(1) == B(1).$

$B + B(1,1) == B(1,1).$

.....

$C + C == C(1).$

$C + C(1) == C(1).$

$C + C(1,1) == C(1,1).$

.....

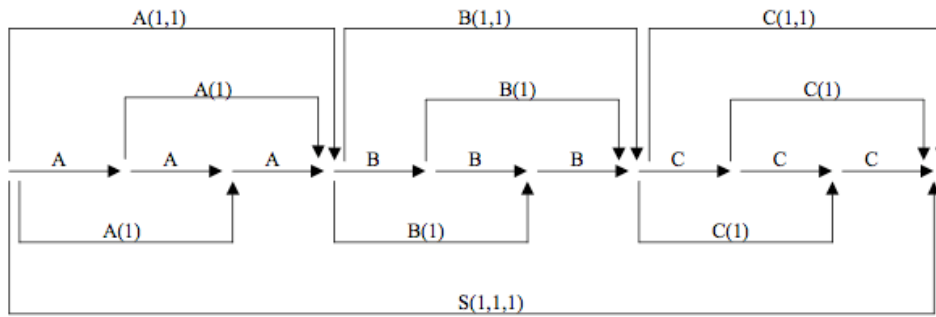
$A + B + C == S(1).$

$A(1) + B(1) + C(1) == S(1,1).$

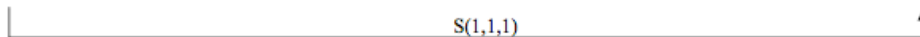
$A(1,1) + B(1,1) + C(1,1) == S(1,1,1).$

.....

On obtient après la première phase :



et après la deuxième :



c'est-à-dire

-0- S(1,1,1) -1-

4.2.2.1.1.3.Exemples de règles utiles

- \$\$

Pour faciliter le développement des traitements-Q, A. Colmerauer a ajouté '\$\$' qui sert à découper une étiquette en une liste de caractères et donc accéder aux éléments qui la composent :

$A^* == MOT(\$A^*)$.

Par exemple, pour l'étiquette 'VERCINGETORIX', cette règle produit :

MOT(V,E,R,C,I,N,G,E,T,O,R,I,X)

- Décomposer un arbre en une chaîne structurées

Les deux règles :

$A^*(U^*) == [(A^*) +](U^*) / U^* -EQ- -NUL- -ET- A^* -HORS- [,]$.
 $](I^*, U^*) == I^* +](U^*)$.

sont utilisées pour décomposer un arbre en une chaîne « structurée ».

Appliquées à

$A(B, C(A, E(F)))$

elles produiraient :

$[(A) + B + [(C) + A + [(E) + F +] +] +]$

4.2.2.1.2.Reconstitution d'un algorithme adapté

4.2.2.1.2.1.Algorithme reconstitué

La syntaxe et la sémantique étant connues, Ch. Boitet nous a proposé un algorithme pour réaliser le « moteur » des systèmes-Q.

C'est naturel de suivre la sémantique décrite par A. Colmerauer qui définit deux phases : addition et suppression.

Dans la phase d'addition, on cherche à appliquer toutes les règles. Avec chaque règle, on va chercher dans le graphe tous les chemins possibles, qui peuvent correspondre à sa partie gauche, et cela en tenant compte des arcs ajoutés par l'exécution de règles.

Sur chaque chemin, on fait l'instanciation de la règle, et on teste si ce chemin correspond à la partie gauche d'une instance de cette règle. Si oui, on évalue la condition avec cette instance. Si la condition est vraie, on ajoute le chemin produit par la partie droite de cette instance entre l'origine et l'extrémité de chemin en question, et on marque que ce chemin a été utilisé.

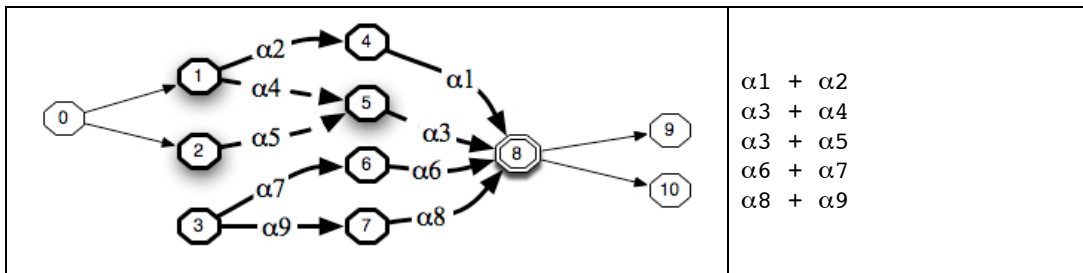
Dans la phase de suppression, comme la sémantique des systèmes-Q, on enlève les arcs utilisés et les arcs qui emmènent à nulle part en partant du nœud entrée pour le nœud sortie du graphe.

L'entrée de l'algorithme est les graphes-Q avec le nœud entrée E et le nœud sortie S, et l'ensemble des règles.

Dans la phase d'addition, on applique les règles de toutes les façons possibles sur le sous-graphe se "terminant" en S. On continue avec tous les nœuds N connectés à S jusqu'au nœud E. On suppose qu'on est en train d'être sur une règle r ayant la partie gauche de longueur l et de la forme :

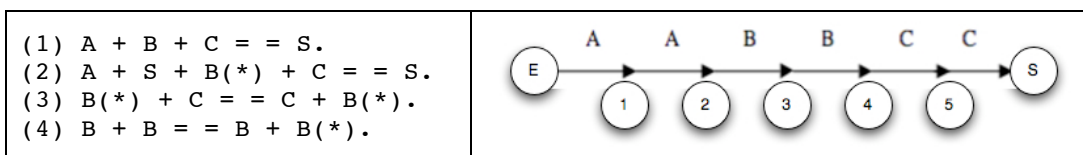
$$A^* + B^* + C^* + \dots == A1^* + F^* + \dots / \text{-CONDITION-}.$$

Selon la sémantique opérationnelle décrite ci-dessus, il faut trouver tous les chemins du graphe correspondant à la partie gauche instanciée de la règle r . On suppose qu'on a exécuté le système-Q jusqu'au nœud N. Pour chaque arc α incident N, on liste tous les chemins possibles qui possèdent comme dernier arc α et sont de longueur l . Dans l'exemple ci-dessous, si N courant est (8), et si la règle a une partie gauche de longueur 2, on listera les chemins suivants dans l'exemple ci-dessous.



Pour chaque chemin de cette liste, on cherche toutes les instanciations possibles de la partie gauche de la règle. Pour chacune, on évalue la condition de la règle. Si elle est satisfaite, on ajoute au graphe le chemin instancié (par les valeurs instanciant les variables), en ajoutant son dernier arc à la fin de la liste d'incidence de N. Cette opération est réalisée pour toutes les règles du traitement-Q courant.

Prenons l'exemple sans variable décrit ci-dessus pour illustrer l'application des règles de l'algorithme.



Règle courante (1), de longueur 3, partie gauche : $A+B+C$
 Nœud courant = (S), arcs d'incident : $B+C+C \rightarrow$ pas de correspondance
 Nœud courant = (5), arcs d'incident : $B+B+C \rightarrow$ pas de correspondance
 Nœud courant = (4), arcs d'incident : $A+B+B \rightarrow$ pas de correspondance
 Nœud courant = (3), arcs d'incident : $A+A+B \rightarrow$ pas de correspondance
 Nœud courant = (2), arcs d'incident : $A+A \rightarrow$ pas de correspondance (trop court)
 Nœud courant = (1), arcs d'incident : $A \rightarrow$ pas de correspondance (trop court)
 Nœud courant = (E), arcs d'incident : \rightarrow pas de correspondance (trop court)
 Règle courante (2), longueur est 3, partie gauche : $A+S+B(*)$
 ...
 Règle courante (3), longueur est 2, partie gauche : $B(*)+C$
 ...
 Règle courante (4), longueur est 2, partie gauche : $B+B$
 Nœud courant = (S), arcs d'incident : $C+C \rightarrow$ pas de correspondance
 Nœud courant = (5), arcs d'incident : $B+C \rightarrow$ pas de correspondance

Nœud courant = (4), arcs d'incident : B+B → correspondant (ajouter le chemin)
 Nœud courant = (3), arcs d'incident : A+B → pas de correspondance
 Nœud courant = (2), arcs d'incident : A+A → pas de correspondance
 ...

Pour l'implémentation, on n'a besoin que des structures de données suivantes :

- ensemble des noeuds, avec indication de l'entrée et de la sortie.
- ensemble des arcs (2 arcs différents peuvent porter 2 arbres égaux).
- un premier booléen ("utilisé") sur chaque arc, et un deuxième ("complété", "accessible") sur chaque nœud.
- une représentation par "listes d'incidence" (on peut ajouter les listes d'adjacence): pour chaque nœud du graphe-Q, on a la liste des arcs entrants.

Voici l'algorithme détaillé du moteur :

```

-----
--- Algorithme général de CB pour le moteur des systèmes-q (9/10/07) ---
-----
EQ (gram, graphe) // la grammaire et le graphe sont en mémoire
{
// le graphe sera modifié (RW), pas la grammaire (RR)
e := entrée (graphe); s := sortie (graphe); // 2 noeuds distingués
marquer e "complété" ; // booléen 'complété'
marquer les autres noeuds "non complété";
marquer chaque arc "non utilisé"; // booléen 'utilisé'
AppliQnd (gram, s); // appliquer les règles de toutes les façons possibles
// sur le sous-graphe se "terminant" en s (ici graphe).
Nettoyer (graphe); // éliminer les arcs ayant servi et les arcs pendants.
}

-----
AppliQnd (gram, nd) // appliquer les règles de toutes les façons possibles
{
// sur le sous-graphe se "terminant" en nd
a := premierInc (nd); // premier arc entrant dans le noeud nd
tantque a <> nil // la liste d'adjacence peut grossir "par la fin"
d := origine(a); // et la terminaison n'est pas garantie !!!
si non (complété(d)) alors AppliQnd (gram, d); fsi // complété: booléen
AppliQarc (gram, a); // tout a été construit jusqu'à d, on peut traiter a.
a := suivantInc (a, nd);
ftantque;
marquer (nd, complété); // booléen "complété" sur le noeud (pour ne pas
refaire!)
}

-----
AppliQarc (gram, ac) // construire tout ce qu'on peut en instanciant le dernier
{
// arc d'une partie gauche de règle avec l'arc ac du graphe
AppliQnd (origine(ac)); // construire tout jusqu'à l'origine de cet arc
pour toute règle r de gram
AppliQrgl (r, ac); // appliquer r de toutes les façons possibles avec ac
// comme dernier arc, ce qui peut ajouter des arcs à Inc(extremite(ac)).
fpour
}

-----
AppliQrgl (r, ac) // appliquer r de toutes les façons possibles avec ac
{
// comme dernier arc.
l := longueur(r, gauche); // longueur du chemin en partie gauche de r
e := extremite(ac);
pour tout chemin c de longueur l et de dernier arc ac // énum. "lexicographique"
d := origine (c); // noeud origine du chemin dans le graphe
pour toute instanciation inst des paramètres de r // énum. "lexicographique"
si evalCond (r, c) // évaluer la condition de la règle
alors
    
```

```

        ajoutChemin (d, e, inst(droite(r))); // ajouter l'instance de la p.d.
        // la liste d'adjacence de e s'allonge à la fin
        marquer (c, utilisé); // chaque arc de c est marqué "utilisé"
    fsi
    fpour
fpour
}
-----
Nettoyer (graphe) // Eliminer les arcs ayant servi et les arcs pendants
{
    // Même technique de parcours, sur place
    e := entrée (graphe); s := sortie (graphe);
    marquer e "accessible" et "complété" // booléens 'complété' et 'accessible'
    marquer les autres noeuds de graphe "non accessible" et "non complété"
    // appel sur le noeud de sortie
    NettoiQnd (s); // ne garder que les arcs "licites" entrant en s
    // sur le sous-graphe se "terminant" en s (ici graph)
    // À la fin, le graphe est vide (cas d'erreur) si et seulement si s est marqué
    non accessible.
    // Ce cas d'erreur est à traiter par l'appelant.
    // Remarque: en Ariane-Y, on prévoira un "code retour" comme le "rc" de REXX.
}
-----
NettoiQnd (nd); // nettoyer entre e et nd
{
    // traiter tous les arcs incidents à (entrant dans) nd
    si complete(nd) alors exit; fsi // nœud déjà traité, inutile de recommencer
    arc := premierInc (nd); // parcours des arcs entrant dans le nœud nd
    tantque a <> nil
        d := origine(arc);
        si utilise(arc) alors effacer (arc); // effacement d'un arc utilisé
        sinon si non accessible(d) alors NettoiQnd (d); fsi
        si accessible(d) alors marquer (nd, accessible) sinon effacer (arc), fsi
        fsi // effacement d'un arc pendant
        arc := suivantInc (arc, nd);
    ftantque;
    marquer (nd, complété); // booléen "complété" sur le noeud (pour ne pas refaire)
}
-----

```

4.2.2.1.2.2. Structure de données

Pour représenter les objets avec un tel algorithme, on n'utilise que des tableaux et on définit la structure de données suivante en C :

Chaîne	<pre> typedef struct { // Caractère dans une chaîne (représentée comme une liste) intsuivant; // caractère suivant intnumero; // ID interne charcar; // valeur du caractère } T_CarChaîne, *Tp_CarChaîne; T_CarChaîne tbCarsChaîne[NbMaxCarsChaîne]; // Tableau des caractères de chaînes </pre>
Arbre	<pre> typedef struct { // Noeud d'arbre paramétré ou arbre constant intfrereD; // frère droit, -1 si dernier fils et benjaminUtilise = 1 intnumero; // ID interne intfilsG; // fils gauche (= fils aîné) intbenjamin; // 1 si dernier fils, 0 sinon -- enum T_NoeudSchematypeNoeud; // E, A, G, I, U intnumPar; // si paramètre, numéro ds tabParam, sinon 0 intR_Chaîne; // valeur de chaîne, indice dans le tableau des chaînes (ref. à tbCarsChaîne) } T_NoeudArbre, *Tp_NoeudArbre; T_NoeudArbre tbNoeudsArbre[NbMaxNoeudsArbre]; </pre>
Traitement	<pre> typedef struct { // Traitement-Q intsuivant; // ref vers ID de traitement-Q suivant intnumero; // ID interne de chaque traitement-Q </pre>

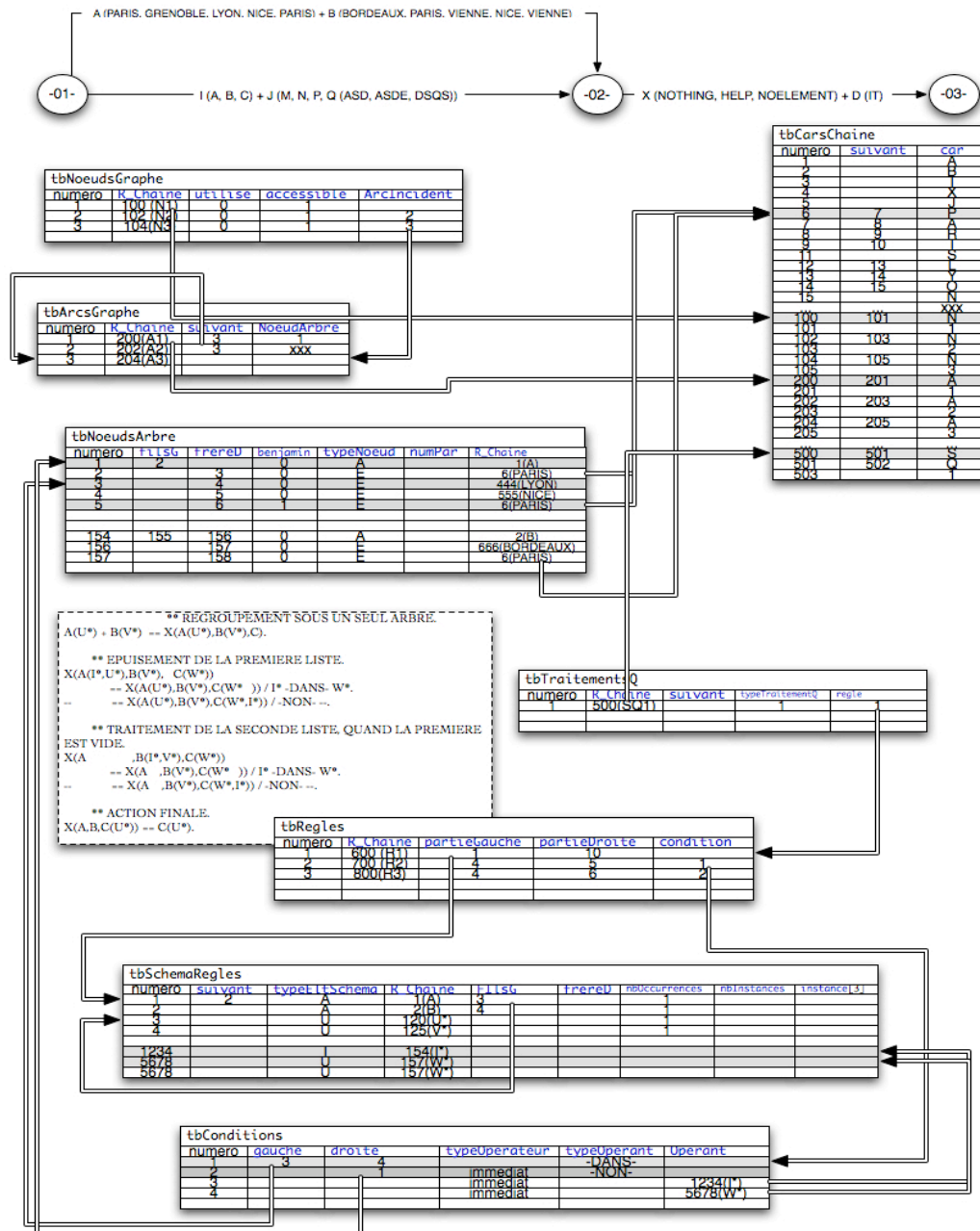
Intervention de non-spécialistes sur les composants logiciels de la TA

	<pre> intR_Chaine; // Nom externe (généré automatique par défaut), réf vers liste des chaînes intregle; // ref vers première règle dans liste des règles de ce traitement (ref. tbRegle) enum T_PhraseTraitementQtypeTraitementQ; // Type de traitement-Q (-DET-, -INV-, -SUP- ou -DIC-) intcommentaire; // Commentaire global d'un traitement-Q intligne; // Numéro de la ligne } T_TraitementQ, *Tp_TraitementQ; T_TraitementQ tbTraitementsQ[NbMaxTraitementsQ]; // Table des traitements-Q </pre>
Règle	<pre> typedef struct { // Règle intnumero; // ID interne de chaque règle intR_Chaine; // nom externe (généré automatique par défaut), ref. vers liste de chaînes intpartieGauche; // Référence vers premier élément de la partie gauche (vers tbNoeudArbres) intpartieDroite; // Référence vers premier élément de la partie droite (vers tbNoeudArbres) intcondition; // Référence vers tbConditions intcommentaire; // Commentaire dédié à une règle intinfo_editeur; // Information externe pour l'éditeur graphique après (pour l'instant non utilisé) intligne; // Numéro de la ligne } T_Regle, *Tp_Regle; T_Regle tbRegles[NbMaxReglesGramr]; </pre>
Condition	<pre> typedef struct { // Condition intnumero; // ID interne intgauche; // ref vers ID élément gauche intdroite; // ref vers ID élément droite enum T_OperateurtypeOperateur; // Type d'opérateur (-ET-, -OU-, -EG-, -HORS-, -DANS-, -NON-, -DIFF-) enum T_OperandetypeOperande; // Type d'opérande (soit 'immédiat' le gauche vers directement à la chaîne concrète, soit 'virgule' les deux gauche et droite vont vers des listes paramétrées) intOperande; // Référence vers liste de chaînes en cours d'instanciation } T_Condition, *Tp_Condition; T_Condition tbConditions[NbMaxConditionsRegle]; </pre>
Graphe	<pre> typedef struct { // Graphe intnumero; // ID interne intR_Chaine; // valeur de chaîne, indice dans le tableau des chaînes (ref. à tbCarsChaîne) intentree; // Entrée de graphe (ref. à TbNoeudsGraphe[]) intsortie; // Sortie de graphe (ref. à TbNoeudsGraphe[]) intnbArcs; // Nombre d'arcs intnbNoeuds; // Nombre de noeuds inttaille; // Taille de graphe intcommentaire; // commentaire dédié pour un graphe } T_Graphe, *Tp_Graphe; T_Graphe tbGraphes[NbMaxGraphe]; </pre>
Arc	<pre> typedef struct { // Arc d'un graphe intnumero; // ID interne intR_Chaine; // nom externe, ref liste des chaînes. intnoeud; // noeud où cet arc commence, ref dans tbNoeudGraphe[] intnoeudArbre; // référence vers liste de Noeuds d'arbre (ref. T_NoeudArbre) intsuivant; // ref. ID d'arc suivant de la liste des arcs entrants pour un noeud; 0 si cet arc est le dernier. enum booleanutilise; // Arc utilisé ou non } T_ArcGraphe, *Tp_ArcGraphe; T_ArcGraphe tbArcsGraphe[NbMaxArcsGraphe]; </pre>
Noeud	<pre> typedef struct { // Noeud d'un graphe intnumero; // ID interne intR_Chaine; // valeur de chaîne, indice dans le tableau des chaînes (ref. à tbCarsChaîne) </pre>

```

enum booleanvisite;// 1: visité; 0: pas encore
enum booleanaccessible; // 1: accessible; 0: non-accessible
intarcIncident;// référence vers liste des arcs incidents (entrants)
intcommentaire;// commentaire dédié à un noeud
} T_NoeudGraphe, *Tp_NoeudGraphe;
T_NoeudGraphe tbNoeudsGraphe[NbMaxNoeudsGraphe]; // Tableau des noeuds de graphe
    
```

Avec une telle structure, on a un exemple de mise en place des données. Notons que les flèches sont bien des références, mais qu'elles sont implémentées par des numéros (entiers) dans les tableaux, et pas par des pointeurs.



4.2.2.2. Points intéressants de l'implémentation reliés aux principes concernés

4.2.2.2.1. Compilation en ANTLR et chargement d'une SI

Nous avons utilisé ANTLR pour compiler les grammaires-Q et les graphes-Q, et les charger dans la structure interne. La grammaire complète en ANTLR est dans l'annexe 3. Dans cette grammaire, on a essayé de ne pas mélanger la description de la syntaxe et l'implémentation directe en utilisant des primitives fournies par la SI.

Pour l'instant, tous les objets sont mis dans des tableaux (voir la structure interne), la taille de ces tableaux est fixée à un très grand nombre d'éléments. Il est prévu d'intégrer le module TabFich pour traiter un nombre illimité d'objets, et permettre de faire des modifications incrémentales.

La conservation des commentaires dans la structure interne est un problème intéressant. Un ou des commentaires peut être attaché à un objet selon sa position. Par exemple :

```
** REGROUPEMENT SOUS UN SEUL ARBRE.  
A(U*) + B(V*) == X(A(U*),B(V*)),C ** CECI EST UN NOUVEL ELEMENT.).
```

Le commentaire ('REGROUPEMENT SOUS UN SEUL ARBRE') peut être attaché à la règle, tandis que ('CECI EST UN NOUVEL ELEMENT') est plutôt attaché à l'étiquette C.

En plus, on devrait éventuellement conserver strictement la présentation des commentaires, car elle porte des informations, par exemple des alignements.

Pour traiter les commentaires, on ne peut pas les introduire dans la grammaire syntaxique, car ils introduisent toujours des ambiguïtés, et la grammaire résultante ne peut donc plus être une grammaire LL(k), pour aucun k. ANTLR propose une solution qui est l'utilisation des canaux de flot lexical. Au lieu de mettre les commentaires dans le canal lexical des caractères non traités comme BLANC, TAB, on déclare un autre canal spécifique pour les commentaires et on les y attache.

```
@lexer::header {  
    #define SYSQ_CHANNEL_COMMENT    1    // 0: DEFAULT, 99: HIDDEN,  
    que DEFAULT sera reconnu par parser, les autres NON  
}  
Commentaire:  
    STAR STAR (~ ( '.' ) ) * '.'  
    { $channel=SYSQ_CHANNEL_COMMENT; }  
    ;
```

En comparant la position des items lexicaux et des commentaires dans le flot d'entrée, on peut déterminer à quel objet syntaxique un commentaire est rattaché, et on peut alors le ranger dans le champ « commentaire » de cet objet.

Avec ANTLR, on a un moyen assez puissant pour traiter les erreurs lexico-syntaxiques. En cours de compilation, on peut capturer l'erreur envoyée par ANTLR et décider de continuer ou d'arrêter le processus de chargement en cours. Par contre, il est difficile de traiter les cas d'erreur sémantique en cours de chargement.

4.2.2.2.2.Moteur

L'implémentation du moteur est faite en C (16 905 lignes de code). On évite le plus possible d'utiliser la récursivité. On n'a utilisé que trois piles (LIFO) pour réaliser l'algorithme décrit. La première sert à parcourir le graphe, la deuxième à l'instanciation et la troisième à l'évaluation des conditions.

4.2.2.2.3.Moniteur simple utilisant REXX

Nous avons implémenté un moniteur multilingue minimal des systèmes-Q. Nous pouvons suivre l'exécution en mettant 4 niveaux de trace selon le détail de présentation souhaité.

Pour implémenter le moniteur interactif à ligne de commande, nous avons utilisé Regina REXX.

4.2.2.3.Évaluation & perspectives

4.2.2.3.1.Test et mesures sur des exemples connus

Les systèmes-Q ont longtemps été utilisés au GETA, dans leur implémentation Fortran venant de l'Université de Montréal. Nous disposons donc de beaucoup d'exemples, parfois avec les traces d'exécution. Nous les avons utilisés comme jeux de test pour valider notre nouvelle implémentation. Voici un exemple avec paramètres. Il s'agit d'un programme qui réalise l'union de deux ensembles d'éléments :

<pre> Grammaires-Q (SYSQFILE.GRUNION.orig) -DET- ** REGROUPEMENT SOUS UN SEUL ARBRE. A(U*) + B(V*) == X(A(U*),B(V*),C). ** EPUISEMENT DE LA PREMIERE LISTE. X(A(I*,U*),B(V*), C(W*)) == X(A(U*),B(V*),C(W*)) / I* -DANS- W*. -- == X(A(U*),B(V*),C(W*,I*)) / -NON- --. ** TRAITEMENT DE LA SECONDE LISTE, QUAND LA PREMIERE EST VIDE. X(A ,B(I*,V*),C(W*)) == X(A ,B(V*),C(W*)) / I* -DANS- W*. -- == X(A ,B(V*),C(W*,I*)) / -NON- ---. ** ACTION FINALE. X(A,B,C(U*)) == C(U*). </pre>
<pre> Données-Q (SYSQFILE.INUNION.orig) -01- A (PARIS, GRENOBLE, LYON, NICE, PARIS) + B (BORDEAUX, PARIS, VIENNE, NICE, VIENNE) -02- </pre>

Voici la trace avec l'analyse :

Commande exécutée :

```

./SysQ ../TEST/DONNEES-Q/SYSQFILE.INUNION.orig ../TEST/TRAIEMENTS-
Q/SYSQFILE.GRUNION.orig TRACE=S
(TRACE=S, la trace ne montre que les règles exécutées avec l'information sur les variables
instanciées)

```

- Première étape

```

** Cette règle a été exécutée en premier sur le graphe-Q pour grouper d'abord les deux
ensembles dans un arbre.

```

```

A(U*) + B(V*) == X(A(U*),B(V*),C).

```

```

** Le chemin A(...) + B(...) correspond à la partie gauche:

```

```

-+1/_1-A(PARIS,GRENOBLE,LYON,NICE,PARIS)-+2/N2-B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE)-
+3/_2-

```

```

** On ajoute un nouveau chemin par la partie droite de la règle:

```

```

-+1/_1- X(A(PARIS,GRENOBLE,LYON,NICE,PARIS),B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE),C)
-+3/_2- /

```

```

** Voici le détail de l'instanciation des variables:

```

```

B: B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE)
U*: PARIS,GRENOBLE,LYON,NICE,PARIS
A: A(PARIS,GRENOBLE,LYON,NICE,PARIS)
V*: BORDEAUX,PARIS,VIENNE,NICE,VIENNE

```

- Suite de sélection des éléments dans le premier ensemble

```

** Cette exécution prend 'PARIS' et le met dans l'ensemble résultat.

```

```

X(A(I*,U*),B(V*),C(W*)) == X(A(U*),B(V*),C(W*,I*)) / -NON- I* -DANS- W*.

```

```

** On ajoute un nouveau chemin :.

```

```

-+1/_1- X(A(GRENOBLE,LYON,NICE,PARIS),B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE),C(PARIS))
-+3/_2- / -NON- PARIS -DANS-

```

```

** Voici le détail de l'instanciation des variables:

```

```

U*: GRENOBLE,LYON,NICE,PARIS
I*: PARIS
B: B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE)
A: A(PARIS,GRENOBLE,LYON,NICE,PARIS)
X: X(A(PARIS,GRENOBLE,LYON,NICE,PARIS),B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE),C)

```



```

V*: BORDEAUX,PARIS,VIENNE,NICE,VIENNE
C: C
W*:

** Cette exécution prend 'GRENOBLE' et le met dans l'ensemble résultat.
X(A(I*,U*),B(V*),C(W*)) == X(A(U*),B(V*),C(W*,I*)) / -NON- I* -DANS- W*.
** On ajoute un nouveau chemin :.
-+1/_1- X(A(LYON,NICE,PARIS),B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE),C(PARIS,GRENOBLE))
-+3/_2- / -NON- GRENOBLE -DANS- PARIS
** Voici le détail de l'instanciation des variables:
I*: GRENOBLE
V*: BORDEAUX,PARIS,VIENNE,NICE,VIENNE
C: C(PARIS)
B: B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE)
A: A(GRENOBLE,LYON,NICE,PARIS)
X: X(A(GRENOBLE,LYON,NICE,PARIS),B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE),C(PARIS))
W*: PARIS

...
** On continue à mettre des éléments LYON, NICE dans l'ensemble de résultat jusqu'à
'PARIS' de nouveau.
X(A(I*,U*),B(V*),C(W*)) == X(A(U*),B(V*),C(W*)) / I* -DANS- W*.
** On ajoute un nouveau chemin :.
-+1/_1- X(A,B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE),C(PARIS,GRENOBLE,LYON,NICE)) -
+3/_2- / PARIS -DANS- PARIS,GRENOBLE,LYON,NICE
** Voici le détail de l'instanciation des variables:
I*: PARIS
V*: BORDEAUX,PARIS,VIENNE,NICE,VIENNE
C: C(PARIS,GRENOBLE,LYON,NICE)
B: B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE)
A: A(PARIS)
X: X(A(PARIS),B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE),C(PARIS,GRENOBLE,LYON,NICE))
W*: PARIS,GRENOBLE,LYON,NICE

• Suites de sélection des éléments dans le premier ensemble
** Cette règle prend 'BORDEAUX' et le met dans l'ensemble résultat.
X(A,B(I*,V*),C(W*)) == X(A,B(V*),C(W*,I*)) / -NON- I* -DANS- W*.
** On ajoute un nouveau chemin :.
-+1/_1- X(A,B(PARIS,VIENNE,NICE,VIENNE),C(PARIS,GRENOBLE,LYON,NICE,BORDEAUX)) -+3/_2-
/ -NON- BORDEAUX -DANS- PARIS,GRENOBLE,LYON,NICE
V*: PARIS,VIENNE, NICE, VIENNE
I*: BORDEAUX
C: C(PARIS,GRENOBLE,LYON,NICE)
B: B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE)
A: A
X: X(A,B(BORDEAUX,PARIS,VIENNE,NICE,VIENNE),C(PARIS,GRENOBLE,LYON,NICE))
W*: PARIS,GRENOBLE,LYON,NICE

** On continue de la même façon que pour premier ensemble, jusqu'au dernier élément
'VIENNE'.
** La dernière règle regroupe le résultat.
X(A,B,C(U*)) == C(U*).
** On ajoute un nouveau chemin :.
-+1/_1- C(PARIS,GRENOBLE,LYON,NICE,BORDEAUX,VIENNE) -+3/_2- /
** On ajoute un nouveau chemin :.
C: C(PARIS,GRENOBLE,LYON,NICE,BORDEAUX,VIENNE)
B: B
A: A
X: X(A,B,C(PARIS,GRENOBLE,LYON,NICE,BORDEAUX,VIENNE))
U*: PARIS,GRENOBLE,LYON,NICE,BORDEAUX,VIENNE

** Le résultat final:.
-+1/_1- C(PARIS,GRENOBLE,LYON,NICE,BORDEAUX,VIENNE) -+3/_2-

```

4.2.2.3.2. Exemple de l'annotation de textes dans le projet OMNIA

Le projet ANR OMNIA [OMNIA, 2009] a pour but la recherche et l'indexation d'images accompagnées de textes. Les images et les textes accompagnés sont traités séparément. Les résultats sont transformés aux descripteurs correspondant à un domaine d'ontologie Ω , puis on émerge à un descripteur unique dans A-box de Ω .

Dans le traitement, après avoir récupéré le texte de la légende des images, on passe à la phase de lemmatisation via NOOJ [Silberztein, 2009], puis on annote les textes lemmatisés par de lexèmes interlingues, qui sont les UW d'U++C fabriqué à partir de Wordnet [Iraola, 2005].

Les deux étapes suivantes sont : la désambiguïsation par vecteurs conceptuels [Lafourcade, Prince et Schwab, 2002] et l'extraction de contenu aux descripteurs Ω .

Dans la phase d'annotation, la sortie de NOOJ est directement utilisée par le module d'annotation de texte dans le cadre du projet OMNIA [Rouquet et Nguyen, 2009]. Dans une première étape, cette sortie est transformée en graphe-Q par un transducteur écrit en ANTLR. Dans une deuxième étape, les lemmes trouvés sont envoyés à PIVAX, qui produit un dictionnaire lemmes-UW pour ce graphe sous forme d'une suite de « grammaires-Q » dont l'une est en fait un dictionnaire. La troisième étape consiste à appliquer ce dictionnaire au graphe des lemmes, ce qui produit un graphe-Q contenant le graphe des lemmes, et de nouveaux arcs (et arbres) contenant une représentation des UW trouvés.

Par exemple, pour le texte « Iraqi women sitting in a waiting room of Bagdad hospital », la sortie de NOOJ donne le graphe de la Figure 52.

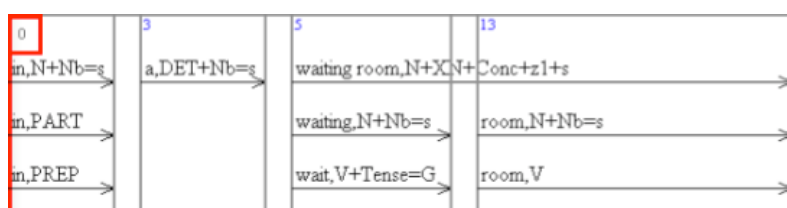


Figure 52: Sortie de NOOJ

On transforme cette sortie en un graphe-Q de lemmes équivalent et on réalise l'annotation par un dictionnaire généré depuis PIVAX. Le résultat de ce traitement est montré dans la Figure 53.

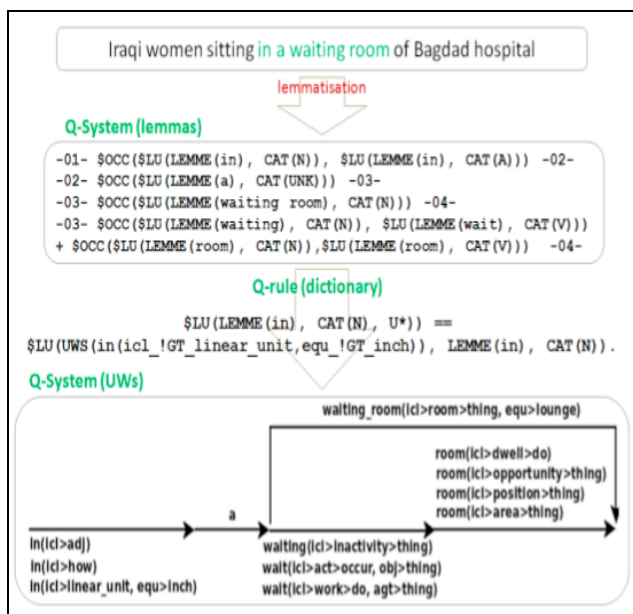


Figure 53: Annotation de texte par les systèmes-Q dans le projet OMNIA

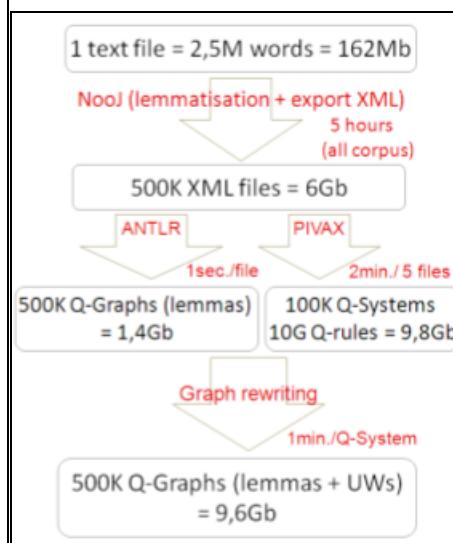


Figure 54: Temps de traitement dans le projet OMNIA

L'expérimentation a été réalisée sur 500K de légendes d'images avec 50 mots/image, donc 2,5M mots au total. Le calcul est fait à la volée sur chaque légende. Le temps de calcul sur un PC (2Go RAM et 2,1GHz) et la taille de données sont présentés dans la Figure 54.

4.2.2.3.3. Variantes possibles

4.2.2.3.3.1. Étiquettes étendues aux caractères Unicode

L'implémentation minimale des systèmes-Q ne supporte pas les caractères Unicode, mais seulement ceux prévus originalement par A. Colmerauer. C'est une des extensions indispensables. Pour la faire, il nous suffit de modifier la grammaire d'ANTLR en ajoutant:

```
UNICODE_CHAR
    : '\\\ ' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
    ;
HEX_DIGIT
    : '0'..'9' | 'a'..'f' | 'A'..'F'
    ;
```

Dans l'espace mémoire interne, on étend la structure des chaînes pour supporter les caractères Unicode. Il suffit pour cela d'utiliser pour chaque caractère sa « valeur Unicode » codée sur exactement 4 octets. En particulier, la longueur d'une chaîne reste proportionnelle au nombre d'octets qui la codent.

On peut aussi utiliser l'encodage UTF-8, mais alors le calcul de la longueur et la comparaison de deux chaînes deviennent en peu plus compliqués.

4.2.2.3.3.2. Variantes du moteur pour le « mode dictionnaire »

Dans une application de TALN utilisant un dictionnaire sous forme de règles, la taille peut être un problème (le dictionnaire du système de TA ATLAS contient 5,57M entrées). Dans l'application de ces règles, il serait utile de marquer que les arcs d'un chemin ont été produits (en tant qu'occurrence d'une partie droite de règle) et de ne pas les utiliser pour la suite, dans des occurrences de parties gauches de règles. On peut introduire la syntaxe (=d=) pour ce type particulier (comme cela a été fait dans le projet TAUM-aviation, 1977-1981). Pour réaliser cela, on ajoute simplement un booléen à chaque arc dans la structure interne d'un graphe-Q, et on interdit à un arc portant ce booléen (à vrai) de participer à une instance de partie gauche de règle.

4.2.2.3.3.3. Compilation incrémentale/efficacité

La compilation incrémentale est un de nos principes importants. Pour l'instant, quand la compilation a une erreur, elle s'arrête et envoie un message au moniteur. L'idée est d'avoir un compilateur plus souple qui demande l'avis de l'utilisateur, et puisse continuer la compilation en passant les erreurs ou en les corrigeant à la volée.

4.3. Aspects liés au moniteur

Comme le moniteur contrôle l'accès aux fonctions du système de TA par des humains, l'étude de la construction d'un moniteur convivial, est utilisable pour utiliser la TA dans un contexte de THAM est un facteur important.

En plus, dans le contexte où l'on a plusieurs systèmes qui communiquent, chacun accède aux autres par la programmabilité donnée par leurs moniteurs.

On va voir ces deux aspects pour la construction d'un moniteur souple utilisable par des personnes et des systèmes.

4.3.1. Idée d'un moniteur ouvert fondé sur un langage de commandes

En partant de l'idée que le système de TA est utilisé par des utilisateurs naïfs, son moniteur doit proposer une interface graphique intuitive. Cette interface doit donner accès à toutes les fonctions possibles du système de TA. Cependant, la présentation d'une telle interface pour des fonctions complexes (désambiguïsation, développement lexical lors de la post-édition, ...) est parfois compliquée, et la simplicité de l'interface pour faciliter l'utilisation cache certaines possibilités du système.

En plus, les besoins des utilisateurs sont très variés. Certains veulent des traitements spécifiques pour leurs applications. Si le système de TA ne dispose que d'une interface graphique et non d'une programmabilité minimum (ex. lancer la TA par des scripts ou des

commandes comme les versions 4, 5 de Systran, mais pas la version 6), on ne peut pas l'utiliser comme il faut.

Il faut donc que le système dispose de commandes primitives et utilisables par les développeurs grâce à des API bien documentées, en mode d'invite de commande. À partir de ces primitives, on peut proposer à des contributeurs extérieurs de construire eux-mêmes des fonctions ou tâches plus sophistiquées.

Du point de vue du génie logiciel, il est toujours vrai qu'il faut ajouter une couche de commandes primitives entre le moteur central et l'interface multimodale. Cette couche permet non seulement d'étendre plus facilement le système, mais augmente sa programmabilité.

De plus, en fournissant un langage de commande enrichi et complet, on peut déléguer l'interface modale à un autre système construit par d'autres pour différents buts (comme CASH par É. Blanc ou WICALE par V. Carpena pour le système Ariane-G5).

4.3.2. Réalisations possibles en utilisant REXX et les REXXstacks

L'avantage de REXX et des piles-files de REXX (REXXstacks) a été discuté au 4.2.1.2.3. Nous les utilisons donc dans l'architecture d'un nouveau moniteur du système Ariane-Y (Figure 55).

Dans cette architecture, on distingue trois couches : les méta-EDL (WICALE, CASH, ...), le centre des traitement des commandes utilisant REXX et REXXstack, avec sa mémoire interne, et le système Ariane existant.

La couche « client » contient des structures internes sous forme physique (fichier), les commandes « primitives » sont envoyées vers un moniteur central. Dans le cas où on voudrait lancer en mode de batch des commandes, il faut analyser (par exemple via ANTLR, et envoyer d'une à l'autre).

Les commandes sont envoyées vers un stack géré par REXX. Un module implémenté sous REXX intervient à la mémoire selon une structure interne bien définie (on pourrait s'inspirer de WICALE, mais en niveau plus profond, car WICALE ne décrit que au niveau module, tous les types d'information (variables, valeurs, formats, dictionnaires, grammaires,... sont considérés comme module). La gestion de la mémoire utilise TabFich. REXX se communique avec ce mémoire via des API de la structure interne.

Il y a une communication entre la mémoire interne et le système de fichiers et la structure existante d'Ariane.

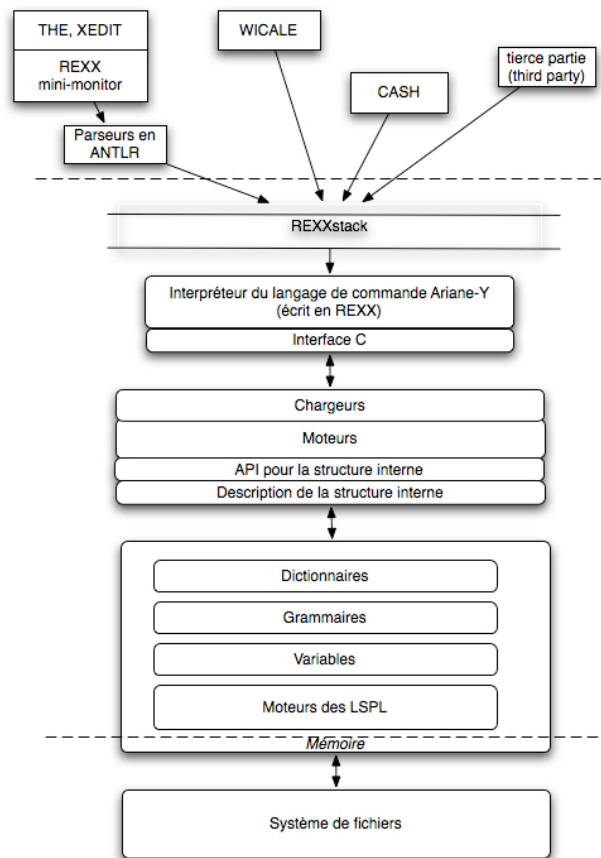


Figure 55: Moniteur d'Ariane-Y en utilisant REXX/REXXstack

Conclusion

Dans ce chapitre, nous avons étudié la construction de systèmes de TAO hétérogènes en ce qui concerne les aspects d'EDL, de l'implémentation des LSPL, et du ou des moniteurs. Nos expériences réalisées ont bien suivi les principes décrits.

Nous avons ainsi dégagé et validé des principes de conception et d'implémentation d'un système de TA extensible et intégrable. Avec l'aspect linguiciel et la mise en service décrits dans les prochains chapitres, on aura une vue globale et complète des aspects concernant la construction d'un système de TAO hétérogène à partir de systèmes de TA et de THAM.

Chapitre 5. Intégration de composants linguiciels pour la TA et pour la THAM

Introduction

En parallèle avec le logiciel, on réalise également la convergence dans la partie linguicielle. La partie la plus importante couvre les connaissances lexicales. La convergence est ici obtenue par la conception, l'implémentation et l'expérimentation de PIVAX, une base de données lexicale multilingue (BDLM) pour les systèmes de TA hétérogènes. La convergence au niveau de corpus est en cours, en appliquant également l'idée d'un système de gestion de corpus unique pour la TA et la THAM. En ce qui concerne la mise en commun des connaissances grammaticales, et encore plus du savoir-faire procédural incorporé aux analyseurs et générateurs, il semble que l'obstacle reste infranchissable, pour des raisons de fond, sauf peut-être au niveau de spécifications formelles ou semi-formelles.

Chaque partie dans ce chapitre présente systématiquement ces composants.

5.1. Base lexicale, intégration de PIVAX à Ariane-Y

5.1.1. Importance primordiale des connaissances lexicales en TA

5.1.1.1. Nécessité pour toutes les techniques de TA

Dans toutes les techniques de TA, les connaissances lexicales sont primordiales et indispensables. Même la TA statistique utilise souvent des dictionnaires en complément des corpus : on considère qu'il s'agit de tables de constituants ou chunk (phrase tables) ou de corpus particuliers (sur lesquels on ne calcule ni alignement ni modèle de langage).

Pour la TA fondée sur les connaissances, après un certain temps, l'effort de développement du système ne se concentre que sur les dictionnaires qui continuent à évoluer, tandis que les grammaires se stabilisent.

5.1.1.2. Application à tous les niveaux lexicaux (mots, idiomes, fragments phraséologiques...)

Les connaissances lexicales concernent non seulement les mots simples, mais aussi les termes, les « idiomes » (expressions), les fragments phraséologiques (*chunks* ou *syntagmes* (*phrase*) en TA statistique). Chaque système ou chaque module du système possède un type de dictionnaire différent selon la nature de son traitement et avec un format précis. Le besoin et l'effort de convergence des connaissances lexicales sont décrits dans la partie 3.

5.1.1.3. Partie majeure du coût de développement

Du point de vue du génie linguiciel, la partie dictionnaire représente la partie majeure du coût de développement linguiciel.

Par exemple, pour un terme français-anglais en aéronautique, cela demande environ 20 minutes de recherche d'équivalent, 10 minutes de codage des propriétés pour la TA (projet B³Vital/aéro/FR).

Le dictionnaire du système de TA ATLAS-II (v.14) contient 5,57M entrées et a été construit depuis 1980 environ avec un groupe de 10 personnes.

5.1.2.Solution pour Ariane-Y : délégation + idée de « serveur lexical » construit comme un agent autonome

Dans le système Ariane-Y, on prévoit d'utiliser une base de connaissance lexicale séparée pour le développement lexical. Cette délégation laisse toutes les fonctions de contribution lexicale à un environnement collaboratif sur le Web. Une telle base fournit aussi des services au système de TA central par l'échange de données.

De plus, une telle base doit pouvoir être utilisée à la fois par plusieurs systèmes de TA. Pour chaque système, on initialise son dictionnaire sur cette base par les dictionnaires existants du système de TA. Comme cette BDLM est utilisée pour plusieurs systèmes avec des dictionnaires différents, on cherche à unifier ces dictionnaires et à compléter les entrées qui existent dans un dictionnaire, mais pas dans un autre. Ces calculs produisent des entrées avec une note relativement basse.

Une BDLM propose aux utilisateurs (lexicographes, linguistes, utilisateurs, bénévoles, ...) de contribuer sur ces dictionnaires, qui sont améliorés au fur à mesure. La modification dans un dictionnaire est envoyée à son système de TA soit en mode actif (sur la période de synchronisation) ou soit en mode passif (quand le système de TA demande à la volée des mots inconnus lors de la traduction).

5.1.3.Nouveaux problèmes et solutions envisagées

5.1.3.1.Problème de délais

Un des problèmes majeurs d'utilisation de la base lexicale est le délai. Un système de TA traduit une page en utilisant ses dictionnaires compilés, chargés en mémoire, et accessibles directement depuis le moteur. Si on utilise une base lexicale, non seulement le chargement des dictionnaires est long, mais la recherche dans les dictionnaires est beaucoup trop lente et affecte nettement la performance de traduction, même si on utilise des mécanismes puissants de système de gestion de bases de données (DBMS) sur les serveurs modernes.

5.1.3.2.Utilisation de « boîtes aux lettres » avec calcul incrémental et boucles infinies

Pour résoudre ce problème de délai, l'idée est d'abord de mettre les données dans un « cache » en mémoire, tout en gardant une base de données pour différentes raisons. Pour un accès, on ne cherche que dans le cache. Pour effectuer les modifications, on change immédiatement les données dans le cache, puis on envoie la commande à la base de données pour demander la mise à jour.

Une autre solution est d'utiliser la technique de « boîte aux lettres ». Il s'agit d'un scénario de communication asynchrone. Quand le système de TA traduit un document, il cherche d'abord dans son propre dictionnaire, puis dans une mémoire tampon où il peut trouver des entrées envoyées par une ou des bases lexicales connectées. Si les mots recherchés n'existent toujours pas, le système de TA envoie une requête aux bases lexicales avec un paramètre de temps d'attente. Il attend la réponse dans la limite du temps d'attente indiqué. S'il y a une base qui a répondu, le système utilise cette entrée. Sinon, il continue la traduction en traitant le mot comme un mot inconnu. Cependant, la recherche de ce mot est maintenue, le résultat arrivera donc dans la boîte de réception après un certain temps. Si de l'information a été trouvée, elle sera utilisée quand le système de traduction rencontrera de nouveau ce mot et mettra à jour son propre dictionnaire.

Ce mécanisme permet de conserver une autonomie forte des systèmes. Et au niveau de la réingénierie, on ne devra pas changer grand chose dans les fonctions et l'implémentation du moteur du système de TA.

Par exemple, si l'analyse morphologique est faite en ATEF, il suffira d'appeler la mise à jour lexicale en cas de mot inconnu, et de relancer en analyse normale. Si le résultat est de nouveau un mot inconnu, on lancera alors le « traitement du mot inconnu ».

5.2. Corpus : intégration de SECTra_w à Ariane-Y

Le corpus est une autre grande partie linguicielle dans le système de TA. L'étude d'un système d'exploitation de corpus « de traductions » ou « pour la traduction » est menée actuellement par HUYNH Cong-Phap au GETALP.

5.2.1. Corpus brut, corpus révisé de TA

La notion de corpus varie selon les systèmes de TA. Pour le processus de traduction, un corpus brut est considéré comme l'entrée du système de TA. Le format de corpus est différent (types de fichiers : PDF, HTML, text ...) avec des fichiers satellite ou non (fichiers image, figure, script, équation, multimédia) et/ou des fichiers compagnon (annotation, graphe d'UNL, dictionnaires, guide de lecture comme TEI, DTD ou Schema XML...).

On cite ici l'exemple du traitement des corpus dans un système de TAO complet écrit en Ariane-G5 pour avoir une vue assez complète des fonctionnalités liées aux corpus dans un système de TA ou de TAO.

Dans le système Ariane-G5, les chaînes doivent être codées en EBCDIC (similaire à l'ASCII). Si l'on veut traiter du chinois ou du japonais, il faut alors utiliser une translittération. D'autre part, il est avéré pratique d'utiliser des transcriptions « minimales » n'utilisant que les 60 caractères de PL/1, en particulier pour éviter les problèmes lors de l'échange de textes. Par exemple, la transcription du français utilisée dans beaucoup de systèmes écrits en Ariane-G5 transforme :

« C'était le Président Nicolas SARKOZY »
en
« *C'E!1TAIT LE *PRE!1SIDENT *NICOLAS **SARKOZY »

Il faut insister sur le fait qu'il s'agit d'un choix des développeurs linguistes, et pas du tout d'une contrainte liée à Ariane-G5 : un caractère en Ariane-G5 peut être n'importe quel octet (de 0 à 255).

Pendant la traduction, un texte est traité en plusieurs phases. Un résultat intermédiaire est produit après chaque phase. Cela est très pratique dans les systèmes multicibles dont le résultat d'analyse dans une langue source est utilisé plusieurs fois afin de le traduire dans plusieurs langues cible.

Après la traduction sur un couple de langues, une unité de traduction (UT) source peut correspondre à plusieurs UT traduites correspondant aux chaînes de traduction. L'utilisateur révise ou post-édite ces UT traduites. La transcription dans le corpus de révision est faite dans un format « intermédiaire », qui est plus proche de l'écriture usuelle de la langue. Par exemple, pour le français on utilise (é) au lieu de (E!1), mais pour le russe, on écrira Pekheshq au lieu de *PEKHESHQ (Печѣшь).

5.2.2. Solution pour Ariane-Y : délégation + idée de « serveur corporal »

Bien que le système Ariane-G5 ait déjà une fonction de post-édition et de gestion de corpus, on voudrait déléguer à un « serveur corporal » le soin de gérer les corpus. La motivation de cette délégation est identique à celle des connaissances lexicales où on pourrait profiter des avantages d'un système sur le Web comme SECTra_w, et surtout avancer vers une unification des ressources corporales dans le contexte des systèmes de TAO hétérogènes.

5.2.3. Nouveaux problèmes

5.2.3.1. Hétérogénéité comme pour la base lexicale

Un des grands problèmes est la diversité de données en théorie ainsi qu'en pratique, comme le montre par exemple le tableau suivant :

Domaine	Organisation	Exemple
Littéral	Ensemble de documents homogènes	Livre, Journal
TA experte	Ensemble des corpus de test ou de production	Systran : unité de test, corpus révisé pour chaque projet dédié. Un corpus est géré comme une partie de code source par un système de gestion de versions (CVS) Ariane : corpus de test, corpus de révision dans divers projets (B'VITAL), corpus intermédiaire de traduction. Les corpus sont gérés via des commandes d'Ariane.
TA experte	Ensemble de textes	1986 bulletins du Referativnyij Zhurnal du VINITI
TA probabiliste	Ensemble des unités de traduction alignées	BTEC, IWSLT-06
MT génération 1	Ensemble des segments alignés	TMX, TBX, UTX...
MT génération 2	Ensemble des sous-segments (chunks)	Treillis étagés dans TELA TransTree

La structure interne des corpus est également hétérogène.

Corpus	Vue logique	Organisation physique	Organisation interne dans un corpus
Ariane-G5 brut	Corpus/Texte	Fichiers « à plat » avec convention de nommage	Liste des unités de traduction (titres, phrases, paragraphes ou texte complet)
Ariane-G5 intermédiaire	Fabriquée selon les phases d'Ariane	Fichiers « à plat » avec convention de nommage liée à la chaîne d'exécution	Une unité de traduction avec un « arbre décoré »
IWSLT-06	Selon le couple de langues, il y a trois types : développement, entraînement et test	Selon la vue logique dans la hiérarchie des fichiers.	Listes de paires d'énoncés <SSeg_L1, TSeg_L2>, chaque énoncé sur une ligne identifiée par un identificateur
BTEC	163 000 multisegments	163 fichiers par langue (18 langues) en codage EUC.	L'entrée contient des segments ou supersegments avec séparateurs de segments, transcrits en minuscules sans

			ponctuations, les listes de segments avec leurs scores
Moses	Suite de segments monolingues	Fichiers texte aligné monolingue	Chaque segment est sur une ligne.
IBM TM	Organisé par paires de langues (anglais vers 25 langues)	Base de données	<anglais, français_i> : liste des occurrences pour cette traduction
ERIM	Organisé par tour de parole selon la dialogue	Fichiers avec description	Un fichier son (.wav) attaché à un fichier de l'annotation selon une convention de nommage

5.2.3.2. Difficulté de concevoir une structure générique

À cause de la diversité des structures et des fonctions, on n'a pas encore actuellement une plate-forme générique pour gérer les corpus de TA et les corpus en général.

On envisage de suivre la solution de Jibiki et de proposer un langage de description de macro et microstructure corporale. Si l'on considère qu'un corpus est similaire à un dictionnaire, un document est similaire à un volume de dictionnaire, et chaque segment correspond à une entrée dans un volume. On peut définir la liaison d'un segment source vers plusieurs segments cible (traduits par un humain ou par un système de MT) via un volume de segments pivot. La conception et la structure à l'intérieur d'un segment est alors décrite dans une microstructure. Cette microstructure peut contenir autant d'attributs qu'on veut pour chaque segment.

Par contre, la solution de Jibiki ne répond pas à deux questions :

- Présentation de la liaison interne des segments dans un même document : Conceptuellement, il n'existe pas de liaison entre deux entrées dans un même volume. Cependant, en pratique on peut facilement étendre par l'ajout d'une entrée dans un volume pivot étiqueté « liaison interne » pour relier deux segments dans un même document.
- Présentation du contexte : on considère qu'un contexte est une vue sur la traduction. Donc, un segment source peut être traduit différemment en deux ou plusieurs segments cible selon le contexte.

5.2.3.3. Passage à l'échelle

Un corpus a normalement une taille plus grande qu'un dictionnaire (le corpus EuroParl représente 44B mots/langue soit 783Mo), surtout avec les corpus multimédia (le corpus TREC Vid⁴⁷ présente 200h d'enregistrement de vidéo avec annotation, corpus ESTER⁴⁸ représente 60h de parole avec transcription orthographique et annotation et 1 700 h non transcrites, la taille totale des corpus sur le marché de LDC (<http://www ldc.upenn.edu/>) est d'environ 650 Go, le plus grand corpus est distribué sur 13 DVD). Donc, il faut gérer le problème de performance et de la limite de taille dans l'algorithme de calcul ainsi que dans le stockage.

⁴⁷ <http://www-nlpir.nist.gov/projects/tv2009/>, visité en août 2009.

⁴⁸ http://catalog.elra.info/product_info.php?products_id=999, visité en août 2009.

5.3. Difficultés conceptuelles et pratiques au niveau des grammaires et des automates

Par rapport aux dictionnaires et aux corpus, les grammaires et les automates sont des composants ayant un formalisme plus compliqué dans les systèmes de TA. Ils dépendent de l'architecture computationnelle et seuls des linguistes spécialisés (« développeurs linguistes ») sont capables de les développer, et il leur faut pour cela un environnement spécifique.

Pour ces raisons, il y a eu certain nombre d'efforts visant à partager des grammaires dans différentes applications de TALN, comme chez Microsoft Research [Reutter et al., 1997] ou entre des langues d'un même système de TA de parole, comme MedSLT [Santaholma, 2007]. Mais il n'y a aucun système permettant de gérer les connaissances grammaticales en commun, ne serait-ce que pour deux systèmes de TA différents (comme SYSTRAN et Reverso par exemple). Il y a des niveaux conceptuels et pratiques pour lesquels on ne peut pas avoir une telle base grammaticale universelle comme pour les dictionnaires et les corpus.

5.3.1. Niveaux conceptuels

5.3.1.1. Particularité dans les grammaires

L'idée d'avoir une grammaire commune pour les systèmes de TA a été proposée depuis 1984 [Kay, 1984]. Théoriquement, on peut concevoir un formalisme de grammaire de « bonne formation » (well-formedness)⁴⁹, qui serait utilisé par plusieurs algorithmes et modèles de calcul. Cependant, les développeurs des systèmes de TA ne l'ont pas suivi. Sans stratégie de choix, on est en effet conduit à calculer toutes les analyses possibles, souvent en nombre exponentiel ($O(2^n)$) par rapport à la longueur n de l'entrée.

Pour aller plus vite, les développeurs ont ajouté des fonctions de contrôle, ou de conditions sous forme des attributs avec les valeurs afin de filtrer ou et de raccourcir les calculs, cela implique les spécifications dans le formalisme au départ. Par exemple, le système ATN (Toshiba, ProMT, Softissimo) permet aux linguistes de définir des contrôles par des variables booléennes pour sélectionner le parcours, le parseur REZO [Stewart, 1975] développé à Montréal a un algorithme universel, mais on définit en réalité des actions et conditions pour paramétrer cet algorithme.

5.3.1.2. Différents algorithmes pour un même modèle

Les règles de grammaire d'un système (ou d'une phase d'un système) sont définies sous une certaine syntaxe. Même si certains systèmes utilisent le même langage ou le même formalisme, avec plusieurs syntaxes, ou même une seule syntaxe, elle est interprétée et exécutée différemment par les algorithmes implémentés. On n'obtient donc pas le même résultat en appliquant 2 modèles différents aux mêmes données linguistiques.

5.3.1.3. Transformation entre les grammaires

Malgré les difficultés de formalismes, il y a eu des efforts visant à convertir une grammaire d'un système dans un autre, comme cela se fait pour les langages de programmation (ex. convertir en Pascal un programme écrit en C). Cependant, ils n'ont pas connu de succès dans le cas des grammaires dynamiques complexes en général. Par contre, on a des possibilités pour les grammaires statiques.

⁴⁹ Par opposition aux grammaires procédurale comme les grammaires de transformation de Chomsky.

5.3.1.3.1. Possibilité pour les grammaires « dynamiques »

Au GETA, nous avons eu deux projets pratique visant à convertir les grammaires d'un système à l'autre. Le premier (1984-85) était dans le cadre des deux projets ETS-4 et ETS-5 avec Eurotra [Lau, 1987] pour convertir des programmes d'ATEF et de ROBRA en des programmes dans le « framework » d'Eurotra. Le second (1998) visait à convertir les programmes écrit en ROBRA d'Ariane-G5 vers TELES (J. Chauché, LIRMM).

Ces projets n'ont pas connu de succès, car il y a toujours trop de différences entre deux langages de ce genre, et on ne sait souvent pas modéliser certaines parties d'un langage par l'autre.

5.3.1.3.2. Possibilité pour les grammaires statiques

Les « grammaires statiques » sont des « grammaires de bonne formation » ou encore des « grammaires de contrainte ». On peut ranger dans cette classe les grammaires syntagmatiques usuelles (linaires, hors-contextes, sous-contextes) et les grammaires d'unification, mais pas les grammaires attribuées, qui font intervenir des procédures de calcul d'attributs. Certaines grammaires dites d'unification (comme les FUG de M. Kay) cachent aussi un aspect procédural (le *ANY des FUG).

Pour la TA, le défaut de ces grammaires est double : (1) elles ne peuvent pas exprimer les correspondances entre, par exemple, des chaînes de caractères et des arbres abstraits, ne serait-ce que parce que ces correspondances sont souvent non projectives ; (2) elles ne permettent pas exprimer de stratégie de choix, en analyse comme en génération. Les « grammaires statiques » introduites par B. Vauquois et S. Chappuy vers 1980 sont une réponse à ce manque.

L'idée de base de grammaires statiques est de décrire des correspondances chaîne-arbre grâce à un formalisme de type hors-contexte avec unification. Ainsi, toute correspondance $\langle c, a \rangle$ sera construite à partir de correspondances $\langle c_1, a_1 \rangle \dots \langle c_p, a_p \rangle$ plus petites, jusqu'à arriver à des correspondances élémentaires entre sous-chaînes et sous-arbres terminaux. Les sous-chaînes considérées peuvent être des sous-mots, c'est-à-dire ne pas être connexes dans la chaîne c de départ. Si $c = m_1 \dots m_n$ a n mots, une sous-chaîne non connexe $d = m_i \dots m_{i+k} m_j \dots m_{j+1}$ formée de 2 sous-chaînes (non vides) pourra être notée par des intervalles dans c , de gauche à droite :

$d = c[(i-1) _ (i+k) + (j-1) _ (j+)]$, avec $1 \leq i \leq i+k \leq j \leq j+1 < n$.

Une GSCS (Grammaire Statique de Correspondances Structurées) aura des règles de la forme $(A, \alpha) \rightarrow (X_1, \beta_1) \dots (X_k, \beta_k) / \text{contrainte}$, où $A_1, X_1 \dots X_k$ désignent des arbres et $\alpha, \beta_1 \dots \beta_k$ des sous-chaînes connexes ou non.

Les GSCS ont été dès le début augmentées de « règles de préférence » indiquant le choix à faire en analyse. Une telle règle est de forme (A, B, γ, C) . Si deux arbres A et B sont en correspondance avec la sous-chaîne γ , on donne la préférence à C . C peut être A ou B , ou encore un arbre dans lequel certaines variables « stratégiques » indiquent l'ambiguïté caractérisée par la double possibilité (A, B) et le choix qui a été fait. Par exemple, GN1 GN2 peut donner lieu à A (dans lequel GN1 est sujet) ou B (dans lequel GN2 est sujet). C aura GN1 comme sujet, ainsi qu'une variable DOUTE contenant la valeur AMBSUJOB. En 1986, la GSCS du français utilisée pour construire l'analyseur du système B'Vital/aéro/FE contenait 150 règles de construction et environ 300 règles de préférence [Boitet, 1986].

5.3.1.3.2.1. Spécification semi-formelle

Au GETA, on a fait une spécification monolingue par grammaires statiques pour l'anglais (ENG), le français (FRA), le malais (MAL), le thaï (THA), le russe (RUS), et l'allemand (GER). Elles ont été utilisées pour écrire les analyseurs et les générateurs des couples de langues: ENG-MAL (anglais-malais), ENG-THA (anglais-thaï), FRA-ENG (anglais-français,

B'Vital). Le système RUS-FRA et l'analyseur de l'allemand avait déjà été implémentés, les GSCS afférentes ont donc servi de documentation supplémentaire.

Les GSCS ont été formalisées, mais la réalisation d'analyseurs et de générateurs à partir d'elles est restée manuelle, bien que systématique. À Penang (USM), Tang Enya Kong a construit pour sa thèse [Tang, 1994] des programmes convertissant une GSCS en un squelette d'analyseur ou de générateur, suivant une stratégie fixée à l'avance.

5.3.1.3.2.2. Spécification formelle

Des spécifications formelles ont été proposées : thèse de Zaharin (1986-87), thèse de R.Zajac (1987), et thèse d'Y. Lepage (1989) sur les Grammaires Statiques d'Indentification. Malheureusement, pour arriver à une sémantique totalement précise, ces chercheurs ont dû supprimer des possibilités importantes des grammaires statiques dans cette conversion : nœuds facultatifs, désordre.

En parallèle, à Penang, il y a eu des efforts pour modéliser les grammaires statiques comme SSTC (Structured String-Tree Correspondence) de Tang Enya Kong, qui a expérimenté la génération de squelettes d'analyse et de génération en ROBRA d'Ariane et en GWS (Grammar Writing System) du système Tapestry [Tong, 1990].

Enfin, Tang Enya Kong et Mosleh Al-Adhaileh ont travaillé sur les S-SSTC pour construire un système de TA par l'exemple. Dans ce système, on n'a plus de grammaire, mais seulement des exemples.

5.3.1.3.2.3. Conclusion sur les informations partageables

À cause des incompatibilités entre les grammaires des systèmes de TA, on ne peut mettre en commun que :

- les espaces de noms : noms et types d'attributs
- les spécifications non procédurales de correspondances entre structures
- les instances de correspondances (S-SSTC) à partir d'annotations sous forme de S-SSTC (correspondances chaîne-arbre alignées)

Les autres informations sont trop spécifiques, on peut au mieux les considérer comme des données brutes.

5.3.2. Niveaux pratiques

Une autre raison pour laquelle les grammaires ne sont pas ouvertes est qu'un changement mineur peut affecter significativement la fonction du système. Donc les systèmes commerciaux n'ont pas intérêt à partager leurs grammaires.

Dans le développement d'un système de TA, les grammaires ne sont pas modifiées autant que les connaissances lexicales. On peut les maintenir pendant certain temps, puis on les garde plus ou moins constantes. Donc, l'intérêt de partager les grammaires est bien moindre que pour les dictionnaires.

5.3.3. Conclusion pour la piste d'une base grammaticale commune

Pour des raisons théoriques et pratiques, nous ne cherchons donc pas à construire une base grammaticale commune entre les systèmes de TA, ni pour une utilisation commune en TA et THAM.

Conclusion

Ce chapitre nous a donné une vue globale sur la réingénierie linguicielle dans le contexte de la construction de systèmes de TAO hétérogènes. La partie de connaissances lexicales a été réalisée, la partie corporale est en cours, tandis que la partie grammaticale reste limitée à un niveau de spécification par grammaires statiques ou par exemples.

Le plus grand problème de la construction d'un système de TAO hétérogène à partir de systèmes de TA et THAM est la diversité des structures de données à tous les niveaux. La solution retenue est de proposer un format et un outil pour décrire les données échangées entre composants des systèmes, et une plate-forme générique pour les gérer. C'est le cas de PIVAX pour la base lexicale et SECTra_w pour le corpus.

On a vu les principes concernés par les architectures computationnelle et linguistique de la réalisation d'un système de TAO hétérogène. Il nous reste à parler des aspects liés à l'architecture opérationnelle où l'on cherche à mettre en service dans un système de TA des fonctions provenant d'un système de THAM.

Chapitre 6. Intégration d'outils et techniques provenant de la THAM

Introduction

L'intégration d'outils et techniques de THAM dans un système de TA est la seconde voie pour intégrer ces technologies.

La première partie du chapitre présente l'intégration de mémoires de traduction (MT) dans la TA. La deuxième partie présente l'utilisation des ressources lexicales de THAM dans la TA. Enfin, la troisième partie montre l'unification de l'interface des fonctions communes entre THAM et TA dans le contexte du Web.

6.1.Problèmes et solutions pour intégrer l'usage de mémoires de traduction à un système de TA

Avec la croissance importante des ressources en MT, l'intégration d'une base de MT dans un système de TA est devenue possible et souhaitable. On voit bien cette tendance dans les versions récentes des systèmes de TA commerciaux, comme SYSTRAN version 7, qui intègrent la recherche dans la MT de l'utilisateur, puis dans les MT disponibles.

L'utilisation de MT dans un système de TA peut être catégorisée en trois grandes stratégies : l'unification de résultat de la TA à la sortie et la recherche dans une MT pour une meilleure qualité, la spécification de la TA en général dans un domaine particulier, et la création d'un système de TA expert à partir de la MT.

6.1.1.Unification entre résultats de TA et de recherche dans une MT intégrée

Pour améliorer la qualité d'un système de TA, on cherche à utiliser les traductions correctes produites lors de la post-édition par les utilisateurs en les proposant de préférence aux TA.

Pour ce faire, on a des méthodes de recherche applicables sur des chaînes pour trouver les chaînes les plus « proches » des chaînes source à traduire.

À la sortie du système, on a une unification entre la TA et la recherche dans la MT. Le résultat peut être multiple et mélangé, ce qui pose le problème de l'identification de l'origine pour améliorer par la suite le système de TA, et de la présentation différenciée des résultats de PE (post-édition) et de TA (brute).

6.1.1.1.Méthodes de recherche dans les MT

6.1.1.1.1.Méthode exacte

La recherche exacte a été largement appliquée grâce à sa simplicité et à sa performance. Le problème général est de rechercher toutes les occurrences de n chaînes $w_1 \dots w_n$ de longueur $l_1 \dots l_m$ dans une chaîne de longueur N sur un alphabet Σ de taille K . Le détail des algorithmes est dans [Charras et Lecroq, 2004; Stephen, 1994].

6.1.1.1.2.Recherche approchée

Pour mieux utiliser la MT, on utilise la recherche approchée au lieu de la recherche exacte. Le détail des algorithmes de recherche approchée peut être trouvé dans [Navarro et Raffinot, 2008].

Cependant, la recherche approchée est une opération compliquée. Par exemple, le système de gestion de MT Trados [TRADOS, 2008] (qui n'a pas de système de TA intégré) fournit une recherche floue (fuzzy match) sur l'ensemble des chaînes pour mieux exploiter la MT. La

qualité des algorithmes de recherche floue actuelle ne satisfait pas les traducteurs. Par exemple⁵⁰, le résultat de recherche floue de « LEAD DESIGN - » :

Source	Fuzzy score
LEAD PROGRAMMER -	75%
Lead Design	67%

Cette fonction de Trados donne aux utilisateurs une interface simple pour sélectionner le seuil de correspondance. Mais les traducteurs lisent toujours en général les autres possibilités au-dessous de ce seuil et prennent une des meilleures.

Améliorer cette fonction est difficile. Pour l'instant, il n'y a pas d'algorithme connu de la recherche floue qui peut s'effectuer sur des segments arbitraires [Reinke, 1999]. Par contre, on peut utiliser une autre technique plus avancée, qui est la recherche floue sur une structure à plusieurs niveaux (treille de chaînes [Planas, 1998]). Cette amélioration a permis à Similis de trouver 20% de chaînes utiles lors d'un travail de traduction pour la CEE utilisant une MT de 20Go, alors que la méthode de TRADOS en trouve 40 fois moins (0,5%).

Vu la difficulté, la recherche approchée est moins efficace que l'intégration de la TA fondée sur les segments dans la MT.

6.1.1.2.Problème d'unification des résultats et prise en compte des retours des traducteurs

Pour un système de TAO intégrant une recherche dans une MT, il faut choisir en sortie des résultats fournis par la TA ou la MT (avec recherche approchée) ou les combiner.

Le choix entre un résultat et un autre peut être réalisé par une approche statistique comme dans le système Pangloss (voir 2.1.1.1). Il y a également une autre technique, basée sur l'approfondissement de la structure de la langue, comme dans le système ELU [Bouillon, Boesefeldt et Russell, 1992].

Il faut noter que, quand l'utilisateur post-édite un résultat « mélangé » fourni par un tel système hybride, il est plus difficile de reconnaître l'origine de la traduction et donc de calculer le bon feedback à envoyer au bon système. Pour éviter ce problème, il nous faut attacher l'information du système concerné à la sortie.

6.1.1.3.Problèmes de présentation

Le résultat d'un tel système peut être multiple. Le système donne toutes les possibilités provenant de la TA et la MT. Le problème est qu'on ne peut pas demander à l'utilisateur de lire toutes les possibilités parmi 2, 3 ou N sorties pour choisir la meilleure.

Il faut une présentation des résultats sous la forme multiple factorisée décrite au 2.1.2.1.2.

6.1.2.Spécialisation de systèmes de TA généraux

Dans un domaine spécifié ou pour une traduction des langues peu dotées où les ressources sont relativement limitées, la création d'un système de TA est plus cruciale. On étudie dans cette partie des techniques possibles pour adapter ou créer un tel système.

6.1.2.1.Adaptation/Spécification de systèmes « experts »

L'adaptation et la spécialisation de résultat de la TA dans un domaine par une mémoire de traduction ont été expérimentées :

⁵⁰ Exemple extrait du blog « About Translation » :
<http://aboutranslation.blogspot.com/2008/01/shouldnt-trados-programmers-improve.html>

Dans le système ATR [Sumita et Iida, 1991], on utilise une MT comme un module de traitement des cas particuliers ou de filtrage à la sortie d'un système de TA par règles pour des phrases fixées paramétrables comme des nouvelles économiques [Kato et Aizawa, 1990].

Dans un autre contexte, on considère qu'un système de TA par règles est un raisonnement par cas [Collins et Somers, 2003], puis on cherche à adapter le raisonnement à un domaine.

Récemment, cette tendance a été appliquée dans des systèmes de TA commerciaux comme SYSTRAN 7 et METAL à Lucy Software [Lucy-Software, 2009] pour renforcer la fluidité de la traduction.

6.1.2.2. Adaptation en TA probabiliste

6.1.2.2.1. Limites

Une des grandes limites de la TA probabiliste est la nécessité d'un grand corpus bilingue aligné (100-1 000K phrases). Ce n'est pas toujours possible, surtout avec les langues peu dotées ou pour la traduction dans un domaine spécifique.

Un autre problème est l'adaptation une fois que le modèle de langue a été construit. Il est quasiment impossible de modifier le système qui est considéré comme une boîte noire ayant peu de réglages possibles.

6.1.2.2.2. Solutions retenues

6.1.2.2.2.1. Annotation, outil pour valider les annotations associées

Lors de la phase d'apprentissage du système, on peut proposer aux utilisateurs de contribuer à améliorer le modèle par des annotations. Pour ce faire, il faut présenter le modèle de langue appris dans un format intuitif aux utilisateurs et leur proposer un environnement pour annoter. On a vu récemment cet effort par N. Ueffing [Ueffing, Haffari et Sarkar, 2007].

6.1.2.2.2.2. Renvoyer des retours, suivre la méthode de Su Keh Yi

Une autre solution est d'utiliser le feedback renvoyé par l'utilisateur. On améliore le système par ré-apprentissage au fur et à mesure de la prise en compte du chargement de corpus en fonction des feedbacks. Le mécanisme de Su Kei Yi (1.1.4) peut être appliqué dans ce cas.

Cette solution est pratique dans un contexte de contribution collaborative (ex. le projet iMAG) où tout le monde peut faire éventuellement de la post-édition ou de la traduction.

6.1.2.3. Adaptation en TA par l'exemple

6.1.2.3.1. TA par analogie de chaînes

Dans l'étude de la TA par analogie de chaînes, [Lepage et Denoual, 2005] a proposé une méthode pour améliorer la qualité de la sortie par l'ajout de paraphrases et de dictionnaires.

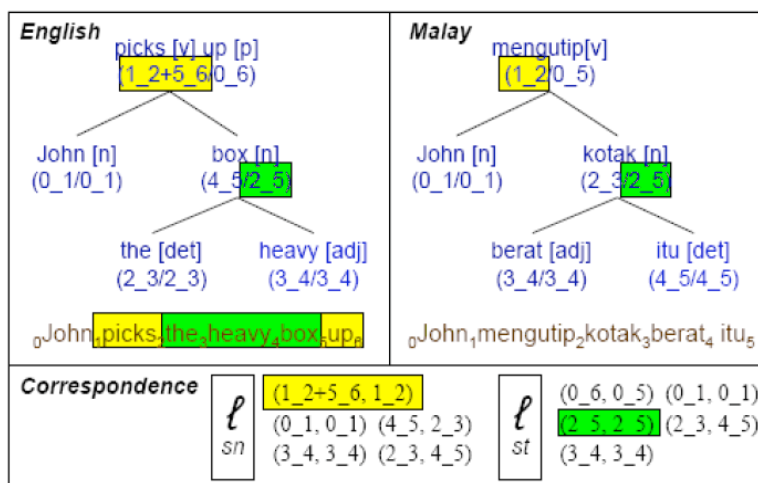
Les paraphrases peuvent être ajoutées en langue source [Yamamoto, 2004], ou en langue cible [Habash, 2002]. Pour ce faire, on groupe les phrases ayant la même traduction (même sens) dans le corpus bilingue. Cette nouvelle information permet de tester un plus grand nombre de proportions analogiques. Quand une paire de phrases (A, B) est proposée pour une phrase d'entrée D, non seulement l'équation $A:B :: x:D$ est testée, mais encore toutes les équations possibles $A':B' :: x:D$ sont testées, où A' et B' sont des paraphrases de A et B.

Le dictionnaire peut être considéré comme une liste d'exemples particuliers. Cela est très pratique quand la phrase source contient des expressions et des numéros.

6.1.2.3.2. TA par alignements structurels

Une solution implémentée dans par le système EBMT à Penang avec les S-SSTC [Al-Adhaileh et Tang, 1999] est d'utiliser l'annotation alignée avec outil associé. Pour un corpus

donné, on fait apprendre par le système, puis on améliore l'annotation de correspondance sur la traduction de chaque phrase, grâce à un environnement.



6.1.3. Création de systèmes de TA spécialisés à partir d'une MT

Quand la TH a produit une ressource de segments bilingues assez grande, on peut créer un système de TA basé sur cette ressource.

Le premier niveau est simple, on cherche des segments similaires du texte d'entrée par les méthodes de recherche approchée. Cependant, pour bien exploiter les ressources, il faut une structure plus sophistiquée que les chaînes à plat comme celle de Similis (**Error! Reference source not found.**). Cela demande un travail important d'annotation humaine sur les segments.

La deuxième approche est d'utiliser un outil (comme Pharaoh, Moses ou Jonas) pour créer un système de TA probabiliste. Cette méthode est facile à réaliser, et on peut construire une nouvelle version du système dès que le corpus bilingue (post-édité) aura assez augmenté.

La troisième est d'utiliser la TA par l'exemple, qui utilise directement les couples de phrases bilingues qui sont dans la mémoire de traduction, sans nécessité de recompilation.

Il existe des systèmes opérationnels basés sur des MT : Pangloss [Brown, 1996]. EDGAR [Carl, 2000].

6.2. Utilisation de dictionnaires de THAM dans des systèmes de TA

Comme les ressources lexicales sont importantes dans la construction d'un système de TA (voir 5.1.1), l'utilisation des ressources lexicales d'un système de THAM dans un système de TA a été mentionnée et essayée avec, par exemple, le couplage entre TM et LMT chez IBM ou d'EuroLang Optimizer avec Logos.

Cependant, il reste encore des problèmes techniques et stratégiques dans cette intégration. Une solution est de construire une base lexicale multilingue pour à la fois THAM et TA. On décrit ici les principes pour la réaliser. Les détails seront donnés dans la partie 3.

6.2.1. Problèmes d'intégration des dictionnaires de THAM et de TA

6.2.1.1. Homogénéité de format

Les connaissances lexicales sont très importantes pour améliorer la qualité, mais elles sont formalisées, et stockées sous forme « codée », pas sous une forme adaptée à un usage général. Donc, c'est un obstacle pour concilier les formats « machine » et « humain » des expressions de connaissances lexicales utilisables dans la traduction.

6.2.1.2.Modification

Les dictionnaires du système de TA sont développés par des linguistes ou des lexicographes, et la post-édition de résultats de TA est plus efficace que l'utilisation de recherche approchée dans la MT (Mémoire de Traductions). Cependant, les dictionnaires des systèmes de THAM sont mis à jour sans cesse par les traducteurs, et ils deviennent de plus en plus volumineux, tandis que les dictionnaires de TA suivent toujours longtemps après (ou pas du tout !) parce que les lexicographes qui s'en occupent ne peuvent pas suivre le développement terminologique et modifier le système de TA assez vite.

6.2.1.3.Compétences des utilisateurs

Pour développer les dictionnaires de TA, ce sont normalement des linguistes, lexicographes ayant une compétence linguistique, et une connaissance suffisante sur le système de TA pour compiler, exécuter, tandis que l'utilisateur d'un système de THAM s'occupe de la contribution lexicale lors de la traduction en utilisant un outil intuitif. Pour inciter les utilisateurs de THAM à développer les connaissances de la TA, il faut qu'on leur propose un environnement de développement lexical facile à utiliser.

6.2.1.4.Opérations

Le développement lexical en TA comporte des opérations spécifiques absentes de la THAM comme l'indexage pour remplir des champs dans le code, la compilation des dictionnaires, la validation croisée... et elles demandent parfois des manipulations informatiques ou des commandes d'un moniteur. Il faut qu'on simplifie l'utilisation de ces outils ou commandes de façon intuitive. Par exemple pour l'indexage, il faut limiter la description linguistique si on veut que les linguistes indexent le dictionnaire (comme le système DUET de Sharp).

6.2.2.Solution retenue de la base lexicale multilingue unique

La solution retenue est d'utiliser une base de données lexicale multilingue (BDLM) unique pour tous les systèmes (plusieurs systèmes de TA, et entre TA et THAM). Il faut extraire les dictionnaires des systèmes de TA et THAM, puis les mettre dans une BDLM centralisée. On met des informations communes et partageables de TA (mot-vedette, terminologie, catégorie morpho-syntaxique, ...) et celles utilisées pour la THAM (peu d'informations grammaticales, peu ou pas de vocabulaire général, des définitions, des exemples).

Pour ce faire, on doit résoudre deux problèmes : (1) créer une BDLM universelle pour un ensemble arbitraire des systèmes de TA, et (2) parvenir à l'utilisation commune entre TA et THAM de cette BDLM. La partie 3 de cette thèse est consacré à résoudre ces problèmes. On ne décrit ici que les principes de réalisation.

6.2.3.Principes de réalisation

6.2.3.1.Construction d'une base lexicale pour la TA et la THAM

La construction d'une BDLM pour un ensemble arbitraire de systèmes de TA est difficile. Quand on a une telle base (chapitre 7), on doit l'étendre pour supporter la contribution lexicale de plusieurs types d'utilisateurs (linguiste, lexicographe, traducteur, utilisateur...).

Du point de vue de l'architecture logicielle, cette BDLM fonctionne comme un serveur lexical pour les systèmes de TA, pour les systèmes de THAM, et pour les humains.

6.2.3.2.Intégration aux systèmes de THAM

Pour renforcer l'utilisation de la BDLM dans le contexte de la THAM, il nous faut l'intégrer dans les systèmes de THAM de post-édition et de traduction. On voudrait déléguer la partie de gestion de ressources lexicales à cette BDLM. Quand l'utilisateur du système de THAM traduit ou post-édite, il peut consulter ou éventuellement contribuer via une interface

simplifiée intégrée dans le système de THAM. Une expérimentation a été réalisée et sera présentée dans le chapitre 8.

6.2.3.3. Offre d'outils de développements lexicaux intuitifs

Le développement lexical d'une BDLM ne doit pas être compliqué pour un utilisateur non-spécialisé. On essaiera de proposer des environnements intuitifs pour que l'utilisateur puisse définir lui-même des tâches primitives. On pourrait alors considérer que le développement lexical est une manipulation directe sur des données lexicales avec une vue graphique. L'étude et le prototypage sont réalisés dans le chapitre 9.

6.3. Unification des interfaces de TH, de post-édition de TA et d'évaluation de TA

En parallèle avec les ressources lexicales, les « ressources corporales » pour la TH (documents traduits, mémoire de traduction) et TA (corpus) peuvent être gérées dans un environnement unique.

Il s'agit d'une recherche menée par d'autres chercheurs au GETALP : Youcef Bey [Bey, 2008], et Cong-Phap Huynh [Huynh, Boitet et Blanchon, 2008]. Dans cette partie, on présente brièvement leurs travaux.

6.3.1. Similarité des interfaces

Il y a une grande similarité entre l'interface et les fonctions associées de la TH, de la post-édition et de l'évaluation de TA. Très souvent, l'interface (voir 1.2.2.1) présente une partie de texte en langue source (à référer), une partie de texte prétraduit (par un ou plusieurs systèmes de TA), et un champ où taper la traduction. Sur cette vue, on peut unifier les fonctions associées de TH et la post-édition ou l'évaluation de TA dans un environnement unique.

6.3.2. Unification déjà en cours

Dans le cadre de sa thèse, Cong-Phap Huynh cherche à unifier la conception et la fonction des systèmes de gestion de corpus (SECTra_w), dont l'interface. Une capture d'interface a été présentée dans (2.3.2.2.3).

Pour utiliser SECTra_w, le corpus ou document à traduire doit être segmenté. Le résultat de la segmentation est une liste de segments et un « fichier squelette », permettant de reconstituer la présentation du fichier en langue cible. On peut passer les segments à un système de TA, puis importer les segments source et les segments prétraduits dans SECTra_w. Sur l'interface, l'utilisateur peut évaluer, post-éditer ou traduire s'il n'y a pas de prétraduction. À la fin, on peut exporter vers une présentation initiale du document en langue cible.

6.3.3. Technique de réalisation à l'époque du Web 2.0

6.3.3.1. Web ou pas Web

La technologie Web évoluant nous permet de construire des systèmes de plus en plus sophistiqués. Cependant, il reste encore des fonctions compliquées non implémentables sur le Web comme l'édition directe ergonomique des graphes ou l'échange d'une grande masse de données qui demande plusieurs minutes de connexion.

	Fonction sophistiquée	Échange de grande quantité de données	Collaboration	Travail hors ligne
Web	-	-	x	-
pas Web	x	x	-	x

6.3.3.2.Utilisation d'un navigateur standard comme base de l'IHM

L'utilisation des navigateurs comme base de l'IHM⁵¹ évite de développer des interfaces spécifiques et permet de profiter de tous les avantages des technologies modernes de programmation Web (comme Ajax, les Wiki, Java) et des outils de génie logiciel associés.

Pourtant, l'utilisation de cette technologie demande de maintenir le système (script, CSS,...) en suivant l'évolution des versions des navigateurs (Internet Explorer, FireFox, Safari) et des systèmes d'exploitation (Windows XP, Vista, 7, Mac OS 9, X ...). Il faut choisir un bon outil, disponible sur toutes les plates-formes, et demandant le moins d'effort possible de maintenance, comme FireFox.

Avec les navigateurs modernes, on peut réaliser des applications avec des interfaces graphiques sophistiquées, mais toujours en format sérialisable, comme XUL de Mozilla.

6.3.3.3.IHM dépouillée (« bare bones ») et fonctionnelle à la Excel et pas d'intégration WYSIWYG

Dans l'interface des systèmes de TH ou de THAM, il vaut mieux adopter une présentation par lignes et colonne qu'une interface compliquée à la WYSIWYG. L'implémentation d'une interface WYSIWYG demande en effet plus d'effort et dépend de bibliothèques logicielles qui risquent toujours de compliquer l'installation et la mise en service pour l'utilisateur non informaticien. De plus, les éléments de présentation et d'ergonomie utiles pour une application donnée, comme la post-édition, risquent toujours d'entrer en conflit avec ceux de la présentation de texte « objet ».

Conclusion

Dans cette partie, on a vu les aspects principaux de la construction d'un système de TA extensible à un système de TAO hétérogène. La construction du méta-EDL générique WICALE et la réingénierie d'un langage spécialisé, les systèmes-Q, ont permis de proposer, d'expérimenter et de valider plusieurs principes de réalisation : langage de commandes pour le moniteur, architecture ouverte pour plusieurs moniteurs, implémentation d'un LSPL en utilisant un outil de génération de compilateur, séparation entre SI (Structure Interne) et SA (Structure Abstraite).

Pour bien comprendre la réingénierie d'un système de TA homogène vers un système de TAO hétérogène, il nous faut maintenant définir et organiser un composant essentiel, une base lexicale utilisable par tous les composants d'un système de TAO hétérogène, C'est-à-dire utilisable à la fois par les humains et par des applications reposant souvent sur des théories lexicales différentes.

⁵¹ Il faut les utiliser de façon à être compatible avec tous.

Partie 3. Gestion des bases de connaissances lexicales et intégration

Introduction

L'intégration de la Traduction Automatique (TA) dans la Traduction Humaine Assistée par la Machine (THAM) est un but incontournable. Cependant, cette intégration n'a pas encore connu de grands succès à cause de raisons techniques et pratiques. Par l'exemple, le système de TA LMT (disponible depuis 1984) et TM/2 (TM actuellement) chez IBM ne sont pas utilisés de façon intégrée parce qu'il y a une séparation entre leurs dictionnaires.

En général, quand on réalise l'intégration entre TA et THAM, les dictionnaires du système de TA sont développés selon les tâches par les linguistes ou lexicographes, et la post-édition de résultats de TA est plus efficace que l'utilisation de recherche approchée dans la MT (Mémoire de Traduction). Cependant, les dictionnaires des systèmes de THAM sont mis à jour sans cesse par les traducteurs, et ils deviennent de plus en plus volumineux, tandis que les dictionnaires de TA suivent toujours longtemps après (ou pas du tout !) parce que les lexicographes qui s'en occupent ne peuvent pas suivre le développement terminologique et modifier le système de TA assez vite. Les traducteurs utilisent alors de moins en moins les prétraductions automatiques, et de plus en plus la MT (qui grossit), et les dictionnaires de TH (qui suivent l'évolution technique et les besoins).

Pour résoudre ce problème, nous proposons d'intégrer les connaissances lexicales pour l'homme et pour la machine dans une base lexicale commune. Ce problème est actuellement aggravé par l'émergence de *systèmes de TAO hétérogènes* dont les composants sont développés par des équipes différentes, à différents endroits, avec des outils différents et surtout avec des représentations différentes des connaissances lexicales.

De tels systèmes ont déjà été construits pour traiter la situation multilingue dans des tâches spécifiques comme la réservation et l'information touristique des projets de traduction de parole comme C-STAR et Nespole!⁵², qui ont utilisé le pivot sémantico-pragmatique (IF, Interface Format).

Pour traiter la langue à grande échelle, l'interlingua UNL est développé depuis 1996-97. Il a maintenant un vocabulaire autonome de 200 000 UW (Universal Words, ou lexèmes interlingues) construit à partir du PWN (Princeton WordNet). Une autre possibilité vient des systèmes de TA monosource et multicible. Dans ces systèmes, les étapes de transfert et de génération peuvent être réalisées par des équipes différentes avec leurs méthodes et connaissances lexicales spécifiques.

Cependant, les systèmes hétérogènes sont souvent créés avec l'idée de « démocratiser » la TA [Boitet, 1999] et d'inciter le public à participer au développement de certaines ressources. Dans tous les types d'architecture computationnelle⁵³ des systèmes de TA, les connaissances lexicales sont très importantes pour la qualité, mais elles sont formalisées, et stockées sous forme « codée », et non sous une forme adaptée à un usage général. Donc, c'est un obstacle pour concilier les formats « machine » et « humain » des expressions de connaissances lexicales utilisables dans la traduction.

Des travaux ont été réalisés pour le but d'échanger les connaissances lexicales entre plusieurs systèmes de TA, comme [Shin-ichiro Kamei, 1999; Shin-Ichiro Kamei et al., 1997] ou le

⁵² <http://nespole.itc.it/>

⁵³ Systèmes « *experts* » se basant sur les connaissances procédurales ou divers types de règles (règles de grammaires ou transitions d'automates), et systèmes « *empiriques* » (SMT, PSMT ou certains EBMT) développés à partir des corpus bilingues.

format OLIF [Lieske, McCormick et Thurmair, 2001]. Mais, pour des raisons exposées au-dessous, ils n'ont pas connu de grands succès. Pour l'intégration entre les dictionnaires de TA et THAM, il n'y a pas beaucoup de précédents à part Yakushite.net [Murata et al., 2003] et le projet DGT-86 [Boitet et Nedobejkine, 1986a].

À partir de ces besoins, nous avons construit PIVAX, une première base de données lexicales multilingue (BDLM) contributive pour briser cette séparation. D'un côté, PIVAX peut être utilisée par des développeurs de systèmes de TA pour (1) gérer les connaissances lexicales dans leurs systèmes sur une plate-forme commune, (2) extraire des dictionnaires de TA dans leur propre format, et (3) agir comme un serveur lexical lors la traduction. De l'autre côté, PIVAX peut être utilisé pour le service de traduction et post-édition collaborative de la même manière que dans le système SECTra_w gère les dictionnaires multilingues et les connaissances « préterminologiques » associées dans divers projets (traduction par une communauté, service d'accès multilingue à l'information, *etc.*). En plus, on veut que PIVAX réagisse comme un EDL où l'on peut définir ou programmer certaines tâches primitives avec un LSPL spécifique pour les informations lexicales.

Le premier chapitre de cette partie explique pourquoi la conception et la réalisation d'une BDLM commune pour les systèmes arbitraires de TA sont très difficiles et propose une solution au problème simplifié concernant des systèmes de TA basés sur un pivot lexical. Le deuxième chapitre décrit l'adaptation et de nouveaux composants pour que PIVAX serve à la fois à la TA et à la THAM. Ensuite, on propose des prototypes d'un langage narratif pour manipuler directement des données lexicales sous la forme de graphes. Enfin, nous décrivons une expérimentation d'un algorithme global pour unifier des ressources lexicales de PIVAX, appliquée au projet U++C.

Chapitre 7. PIVAX, une base lexicale pour les systèmes de TA hétérogènes

Introduction

Dès 1986 [Boitet et Nedobejkine, 1986b], notre équipe de recherche (le GETA à l'époque) a commencé à étudier comment centraliser les connaissances lexicales dans une base de données lexicales multilingue (BDLM). À partir d'une telle BDLM, il est devenu possible dès 1987 de construire automatiquement les dictionnaires monolingues et la partie terminologique des dictionnaires de transfert d'un système de TA écrit en Ariane-G5. La société B'VITAL a utilisé cette technique dans un cadre industriel (système BV/aéro/F-E) avec la base BDTAO.

Du côté industriel, il y a également un besoin de partager des données lexicales. Ce besoin vient du désir d'utilisateurs de systèmes commerciaux comme SYSTRAN, Reverso, METAL, *etc.* de partager leurs dictionnaires entre eux et entre systèmes. On peut citer la plate-forme UPF [Shin-ichiro Kamei, 1999] avec le format UTC-X [Shin-ichiro Kamei et al., 1997] pour l'échange des dictionnaires des systèmes de TA. Cependant, l'information lexicale à partager est limitée à cause de la protection de propriété pour des ressources importantes (comme la terminologie multilingue, et surtout les codes spécifiques de chaque système) qui coûtent cher à développer, donc l'effort de ce côté-là n'a pas tout à fait réussi.

Passant de la TA homogène à la TAO hétérogène, on souhaite avoir une BDLM universelle pour plusieurs systèmes de TA. Mais c'est un objectif très difficile à réaliser. Le plus grand obstacle rencontré par le passé est la diversité des données. L'approche de PIVAX est un compromis entre la généralité et la faisabilité. Nous voudrions dans l'idéal développer une architecture pour une base de données capable de supporter tous les systèmes de TA, avec des architectures linguistiques arbitraires, mais cela semble en soi impossible, parce que les types d'information et leur organisation sont trop différents. On pourrait peut-être y arriver, mais nécessairement de façon extrêmement complexe, parce que des liens devraient être maintenus à plusieurs niveaux (entre les formes de mot, les lemmes, les familles dérivationnelles ou les « prolexèmes », et les lexies).

C'est pourquoi nous restreignons le problème à la classe des systèmes de TA (homogènes ou hétérogènes) utilisant un même « espace lexical pivot ». Actuellement, un nombre croissant de systèmes de TA à « pivot » sont effectivement développés d'une manière hétérogène par des groupes indépendants. Le pivot peut être une des langues naturelles annotées, par exemple l'espéranto dans le projet DLT (1982-89), un interlingua spécifique à une tâche et à un domaine comme l'IF (Interface Format) des systèmes C-STAR-II et Nespole! de traduction de parole, ou un interlingua linguistico-sémantique tel qu'UNL (voir 2.2.3.3). Parmi ces architectures à « pivot », nous nous concentrons sur celles où le « pivot » a son propre « espace lexical » autonome (comme UNL, C-STAR avec l'IF, ou MASTOR-1 [Gao et al., 2006]). Les autres, comme MedSLT [Rayner et al., 2008], ou le premier système du CETA en 1962-71, utilisent des « pivots hybrides » [Vauquois, 1975] où les représentations intermédiaires utilisent des attributs et des relations interlingues et l'espace lexical propre à la langue représentée.

Nous avons conçu et développé PIVAX, le premier système contributif en ligne de base de données lexicales qui permet de créer, maintenir et gérer les ressources lexicales de systèmes de TA basés sur un « pivot lexical », et hétérogènes dans le sens où leurs composants spécifiques à une langue peuvent être développés à différents endroits et avec différentes approches linguistiques et différents outils informatiques. Seule l'information lexicale spécifique à une langue et essentielle pour la traduction doit être stockée dans PIVAX, de sorte que les développeurs peuvent utiliser leurs propres outils et protéger les informations propriétaires.

Dans la section 1, nous présentons l'état de l'art de la gestion des connaissances lexicales dans les systèmes de TA. Dans la section 2, nous rappelons le travail de la construction d'une BDLM et les expériences précédentes, pour comprendre les raisons de leurs limites. Cela nous mène à spécifier l'architecture linguistique de PIVAX (macrostructure et microstructure) et son rôle dans le contexte d'application à un système de TAO hétérogène dans la section 3.

7.1. Gestion des connaissances lexicales en TA homogène

En vue de la construction d'une BDLM commune à plusieurs systèmes, il nous faut avoir une vue assez large sur les dictionnaires des systèmes de TA actuels. On peut les classer selon le degré d'interaction avec le système de TA :

- fichiers à compiler,
- base lexicale séparée et à transformer en fichiers dictionnaires de TA,
- et enfin base lexicale avec consultation et mise à jour lors de l'exécution du système de TA.

7.1.1. Fichiers et BDLex « implicite » à la VISULEX

Usuellement, les gros systèmes de TA utilisent des dictionnaires qui sont contenus dans plusieurs fichiers avec des formats différents (nous avons vu en détail l'exemple d'Ariane-G5). Chaque fichier contient un type d'information, utilisant des codes (abréviations) définis dans des fichiers à part. Par exemple, un système de TA écrit en Ariane-G5 peut avoir jusqu'à 56 fichiers⁵⁴ de dictionnaires, mais, en réalité, cette limite est rarement atteinte. La plupart des couples de langues en ont au plus 15.

Les systèmes commerciaux tels que Softissimo, ATLAS (Fujitsu), The Translator (Toshiba), etc., ont des douzaines de dictionnaires spécialisés à différents domaines. ATLAS-II v.14 a au total 5,57M entrées dans ses dictionnaires spécialisés.

Le système SYSTRAN contient environ 60 fichiers de dictionnaires de bases pour chaque couple de langues. Voici un exemple de la hiérarchie des fichiers pour le couple anglais-arabe (ENAR).

SystranMonolingual (EN):	Homography	Bilingue (ENGLISH-ARABIC)
EN	EN_Table.HM	Transfer
Alignment	Inflection	ENAR
alignment-enar.xml	EN_Table.DETPRO	compoundsenar.txt
TableCodage.ENSQ	EN_Table.NP	possessive_pronouns-enar.lst
TableCodage.ENSJ	EN_Table.V	personal_pronouns-enar.lst
TableCodage.ENAR	EN_Table.N	ordinal_numbers-enar.txt
TableCodage.ENRU	EN_Table.A	transfer-enar.txt
TableCodage.ENPT	Mono	dates-enar.lst
TableCodage.ENPL	te_stopwords-en.lst	negation-enar.lst
TableCodage.ENSJ	Postprocess	frozen_compounds-enar.txt

⁵⁴ 21 dictionnaires d'analyse morphologique (phase AM : 7 pour les morphes usuels et 14 pour les tournures fixées connexes), 14 pour l'expansion lexicale dans l'analyse (phases AX et AY, allant de lemmes à simples unités lexicales, unités lexicales composées et acceptions), 21 en transfert (7 pour le transfert lexical dans TL, 14 pour l'expansion lexicale en langue cible dans TX et TY), et 14 pour la génération morphologique.

TableCodage.ENNL	postprocess_XX-en.lst	
TableCodage.ENJA	Translation	
TableCodage.ENFR	compent-en.lst	
TableCodage.ENZH	vk-en.lst	
TableCodage.ENUK	seg-en.lst	
TableCodage.ENAR	norm_ja-en.lst	
TableCodage.ENKO	norm_cjk_cisco-en.lst	
TableCodage.ENDE	norm_cjk-en.lst	
TableCodage.ENES	norm-en.lst	
TableCodage.ENIT	lookup-en.lst	
TableCodage.ENHU	loca-en.lst	
TableCodage.ENFA	
TableCodage.ENEL		
TableCodage.ENDA		
Guess		

Pour construire une base de données lexicales pour chaque dictionnaire ou pour une classe de dictionnaires spécifiques pour un système de TA, on peut faire la conception à partir de sa structure et réaliser une implémentation directe avec une interface de gestion.

C'est le cas du système VISULEX [Bachut et Verastegui, 1984], un outil de visualisation synthétique de l'information lexicale de tout ou partie d'un système de TA écrit en Ariane-G5. Cet outil permet au lexicographe de voir toutes les informations codées concernant un ensemble d'unités lexicales source (y compris leurs traductions) sur une page, et tous les commentaires associés, sur une page « en regard ». VISULEX est une première approche vers la factorisation des ressources lexicales, mais le format factorisé est très proche de celui du système. Voici un exemple de VISULEX avec l'UL 'CHANGE' (Figure 56) dans l'application de TA anglais-français. Il y a deux niveaux, le premier est à gauche, le deuxième à droite.

7.1.2.1. BDTAO

Pendant son activité (1985-1995), la société de TA B'VITAL est parvenue à gérer avec son outil BDTAO tous les types d'information lexicale, et à en extraire tous les dictionnaires Ariane-G5, sauf les dictionnaires de transfert lexical des mots généraux (non terminologiques) sous forme de dictionnaires Ariane-G5. À l'exécution, on peut demander de régénérer les dictionnaires dans un format prêt à être compilé par Ariane-G5.

Cet outil a en fait été conçu spécifiquement pour la TA, mais indépendamment d'un système particulier. BDTAO était la première base de données lexicale commune à plusieurs systèmes de TA avec l'idée de délégation de gestion des connaissances lexicales dans un système séparé.

7.1.2.2. UWGate

Dans le projet UNL (voir 2.2.3.3), chaque enconvertisseur/déconvertisseur a besoin d'un dictionnaire UW-LN. Afin de maintenir une liste des UW homogènes et unique, l'UNL-C (UNL Center, Tokyo) a développé l'outil UWGate pour accéder à une base centralisée des UW via Internet. Dans cette base, il y a le dictionnaire des UW-LN. Cet outil donne une interface graphique sur Windows à distance (UWGatePlus) pour consulter et éditer des informations sur ces UW pour les utilisateurs inscrits.

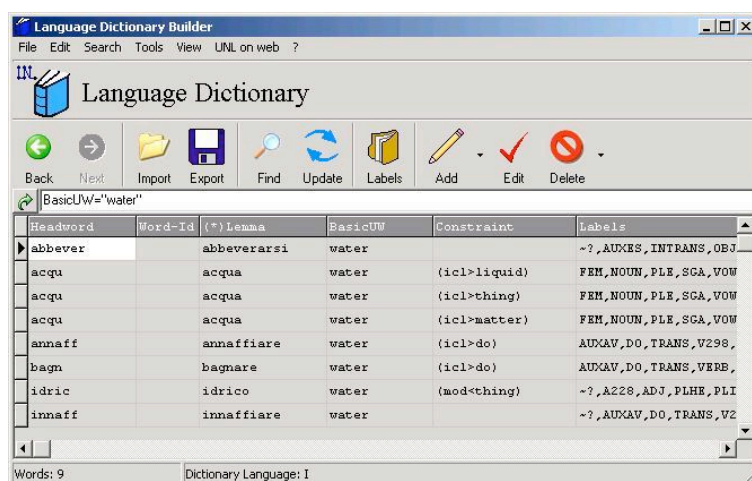


Figure 57: Interface d'UWGate

Dans l'ensemble des outils fournis par UNDL, il y a aussi deux LSPL, EnCo et DeCo, pour créer les enconvertisseurs et déconvertisseurs. À partir de UWGate, on peut exporter le dictionnaire UW-LN vers un fichier ayant le format attendu par EnCo et DeCo.

Cette base UWGate propose une gestion centralisée d'une ressource utilisée par plusieurs composants à distance.

7.1.2.3. PARAX

Cette base de données lexicales multilingue à « acceptions interlingues » a été développée comme un hypertexte (sous HyperCard, puis Revolution) par un linguiste-informaticien, É. Blanc [Blanc, 1999]. Elle comprend un dictionnaire d'UW relié à plusieurs dictionnaires monolingues (actuellement français, japonais, chinois, espagnol et russe). Chaque dictionnaire est une pile de Revolution™. Le lien entre les dictionnaires est réalisé par UNL : chaque mot dans une entrée de dictionnaire monolingue est associé à un UW du dictionnaire construit par l'équipe à Tokyo de H. Uchida en 2002, avec 100 000 entrées. Chaque dictionnaire d'une LN contient de 30K à 60K entrées. Une description morpho-syntaxique complète des entrées est actuellement disponible seulement dans le dictionnaire français. Voici un exemple tiré des dictionnaires UW-FR et UW-ES de PARAX :

```
;; UNL-FR (français)
[biens] {CAT(CATN),GNR(MAS),NUM(PLU)} "goods(icl>functional thing)";
[catalogue] {CAT(CATN),GNR(MAS),N(NC)} "catalog(agt>thing,obj>thing)";
[centaines] {CAT(CATN),N(NP)} "hundreds of";
;; UNL-ES (espagnol)
[necesidad] {VAR} "need(icl<abstract thing)";
[necesidad] {VAR} "need(icl>thing)";
[nivel] {VAR} "level(icl>thing)";
[no ambiguo] {VAR} "unambiguous";
```

PARAX donne une interface en colonnes (Figure 58) pour consulter et éditer les entrées. É. Blanc a intégré PARAX dans CASH (4.1.1.3) permettre faire la contribution lexicale à la volée dans le processus de développement d'un enconvertisseur UNL-FR.

Dans l'organisation, chaque dictionnaire monolingue LN est une pile Hypercard. Il y a un lien direct de chaque entrée dans les dictionnaires LN vers une entrée dans le dictionnaire des UW, qui représente une acception interlingue.

Après avoir été développé, le dictionnaire doit être envoyé à Ariane pour être compilé avant de lancer l'exécution de la TA. PARAX est une base lexicale utilisée à la fois par la TA et la THAM, mais spécifique actuellement à Ariane-G5.



Figure 58: Interface de consultation en colonnes de PARAX

7.1.3. BDLM avec interaction limitée à la consultation durant la TA

Le second type de BD lexicale pour la TAO est destiné à être utilisé par le système de TA lui-même, durant l'exécution : si le système a besoin d'un article de dictionnaire pour un mot rencontré, il le demande à la BD, qui le lui envoie sous une forme comprimée, qui est décompressée en mémoire avant être utilisé. C'est le cas des systèmes MU, METAL et LMT.

7.1.3.1. MU (Adabase)

Dans le cadre du projet MU [Nagao et Tsujii, 1986] pour développer la TA anglais-japonais, on a construit un dictionnaire (5 000 verbes, adjectifs, adverbes, ... et 70 000 noms, y compris de la terminologie) dans une base relationnelle ADABase-Lisp. Un lexicographe peut développer les informations morpho-syntaxiques et syntaxiques (présentées dans un format LISP) par une interface dédiée. Cette base a été intégrée au système de TA MU, dont le logiciel de base et les LSPL sont également écrits en Lisp.

Lors de la traduction, le système charge d'abord toute la base, mais il pourrait aussi la consulter à la demande et ranger les résultats incrémentalement dans l'espace de travail.

7.1.3.2. METAL

Dans le système METAL créé par l'équipe de J.Slocum au LRC (Linguistics Research Center) de l'Université du Texas, à Austin [Bennett et Slocum, 1985], il y a 200 000 entrées monolingues en anglais et en allemand. On a mis ces entrées dans une base de données sous une forme compacte (zip). L'utilisateur peut développer à la volée des entrées. Quand le système s'exécute, il charge d'abord le petit noyau des articles toujours nécessaires (classes fermées), puis accède à la base incrémentalement pour chaque mot rencontré, et développe en mémoire le ou les articles renvoyés par la base lexicale.

7.1.3.3. LMT

Le système de TA LMT (Logic-programming-based Machine Translation) [McCord, 1989] d'IBM a une base lexicale dédiée LOLA (Linguistic-Oriented Lexical database Approach) [Bläter, Schwall et Storrer, 1992]. Les connaissances lexicales (50K entrées pour le couple de langues allemand-anglais) sont stockées dans une base relationnelle SQL/DS. On peut importer dans la base le dictionnaire au format de LMT, contribuer aux informations par l'interface, et extraire les entrées à compiler pour le système LMT.

Pour améliorer la qualité et quantité d'informations lexicales, on a développé un module, qui peut accéder en temps réel au dictionnaire externe LDB et convertir les informations récupérées dans la base LOLA.

LOLA a été initialement construit pour la traduction allemand-anglais. On voulait réutiliser des informations lexicales, pour développer les couples de langues anglais→danois et danois→anglais. Pour cela, on a importé des dictionnaires récupérés anglais↔danois de l'extérieur et on a réalisé des algorithmes pour générer automatiquement la version 0 de ces deux dictionnaires anglais-danois de LMT, en complétant certaines informations automatiquement. Ces dictionnaires sont proposés aux linguistes à corriger, puis utilisés dans la traduction anglais-danois dans LMT.

LOLA de LMT nous donne des idées importantes sur les opérations sur l'information lexicale d'une BDLM : la gestion, la récupération, le développement par des humains et la programmation pour enrichir les informations lexicales.

7.1.4. BDLM avec consultation et édition durant la TA : Pensée et yakushite.net (Oki)

Le système Pensée [Shimohata et al., 1999] a une base lexicale sur le Web (<http://www.yakushite.net/>) [Murata et al., 2003]. Tout le monde peut contribuer au dictionnaire en collaboration via une interface Web. La contribution est révisée et intégrée directement dans le système de TA Pensée.

Lors de la traduction, l'utilisateur peut modifier les entrées dans la base en mode de débogage de traduction de Pensée avec une interface graphique dédiée à Pensée.

C'est le seul cas que nous connaissons où l'on peut modifier incrémentalement le dictionnaire de TA (via la BD lexicale) lors de la traduction.

Voici un exemple d'un dictionnaire créé et géré en ligne par une communauté de traducteurs :

分類：全て（登録順） このコミュニティの分類はありません。

英語見出し	英語品詞	翻訳方向	日本語見出し	日本語品詞	ニックネーム	日付	分類	用語説明
fibroid	名詞	<-->	類線維腫	名詞	aimaimi09	2009/07/27		
skin preparation	名詞	-->	皮膚組織標本	名詞	aimaimi09	2009/07/27		
croscarmellose	名詞	<-->	クロスカルメロース	名詞	aimaimi09	2009/07/27		
croscarmellose sodium	名詞	<-->	クロスカルメロースナトリウム	名詞	aimaimi09	2009/07/27		
Yellow No. [Num]	名詞	<-->	食用黄色[Num]号	名詞	aimaimi09	2009/07/27		
Yellow No. [Num] Lake	名詞	<-->	食用黄色[Num]号レーキ	名詞	aimaimi09	2009/07/27		
Pfizer Pharmaceuticals	名詞	<-->	ファイザー製薬	名詞	aimaimi09	2009/07/27	組織名	
Pharmacia Pharmaceuticals	名詞	<-->	ファルマシア製薬	名詞	aimaimi09	2009/07/27	組織名	

Figure 59: Exemple de dictionnaire de Yakushite.net

7.2. Problèmes posés par l'élargissement à un ensemble de systèmes arbitraires de TA

Pour construire une BDLM spécifique à un système, on peut définir une structure figée spécifique de ce système, donc il est relativement facile de faire la conception et l'implémentation des fonctions de gestion de la base, ainsi que des fonctions spécifiques du système.

Par contre, le désir de la construction d'une BDLM pour plusieurs systèmes de TA nous demande d'avoir une base commune. Cela pose des problèmes difficiles, théoriques et pratiques, que nous analysons plus en détail ci-dessous.

7.2.1. Variété dans la nature des données

7.2.1.1. Information

Dans les dictionnaires de TA, les types d'information constituant l'espace lexical peuvent concerner les unités suivantes, à des niveaux de plus en plus abstraits : formes, morphes (affixes, infixes), lemmes, sens de mots (lexies simples), mots ou termes composés, lexies complexes ou simples associées, familles dérivationnelles (unité lexicale ou UL⁵⁵ des systèmes fondés sur la théorie de Mel'tchuk), prolexèmes [Tran, 2006], lexèmes interlingues (UW d'UNL), et concepts (ex. ontologie de KBMT-89). Cette diversité est illustrée dans le tableau suivant.

Système	Unités utilisées
Ariane-G5	morphe, lemme, UL, UL + sens [numéro]
LMT, METAL	forme, lemme
KBMT-89, KANT	lemme, sens
ATLAS-II, PIVOT, UNL	lexème interlingue
ETAP-3	morphe, lemme, UL, sens [id]
MU	forme, lemme, sens (implicite)
SYSTRAN	forme, lemme (terme, idiome), UL (restreint)

7.2.1.2. Structure

Comme chaque système a été conçu avec une approche lexicale définie en fonction de ses architectures linguistiques et informatiques, on a une grande hétérogénéité de l'organisation des dictionnaires au niveau des macrostructures⁵⁶ et des microstructures⁵⁷ [Polguère, 2004] ou [Sérasset, 1994], et des syntaxes. Voici des exemples :

⁵⁵ On réserve cette notion d'UL pour parler des systèmes de TA fondés sur des dérivations permettant des paraphrases en traduction.

⁵⁶ Macrostructure : organisation des « volumes » ou ensembles d'articles, classiques ou non (ex : volumes et liens comme les axèmes et les axes en PIVAX).

- Macrostructures :

Système	Macrostructure
Ariane-G5	Machine → mini-disque → L1-L2 → étape → phase → Dictionnaire (texte)
SYSTRAN	L1-L2 → Type de dictionnaires → Dictionnaire (texte)
CICC/EDR	Type de dictionnaire → L1-L2 → Dictionnaire (texte)
CAT2	Phase → L1-L2 → Dictionnaire (texte)

- Microstructures :

Système	Microstructure
Systèmes écrits en Ariane-G5	lemme et UL (famille dérivationnelle), ex.. REPAIR_V = {repair_V, repair_VNres, reparation_Nact, reparable_PPA, reparably_PPAdj, repaier_Nagt}. Mais, dans le projet B'Vital, pour faciliter le développement lexical par lexicographe, on a séparé les noms d'agent ou d'instrument et on leur a affecté des UNL différentes, parce qu'ils sont souvent de nature terminologique.
ETAP-3	lemme ou UL plus riche (fonctions lexico-sémantiques en plus)
UW d'UNL	lemme simple ou complexe en anglais avec un ensemble de restrictions, ex. : river(icl>stream,gol>stream) for "rivière" en français river(icl>stream,gol>see) for "fleuve" en français

- Syntaxes :

Système	Syntaxe
Ariane-G5	Tous les dictionnaires sont associés à un des 3 LSPL (ATEF, TRANSF/EXPANS, SYGMOR) avec leurs propres organisations et syntaxes.
SYSTRAN	Formats différents pour chaque dictionnaire (6 types pour un dictionnaire monolingue: <i>Mono, Inflection, Homography, Extraction, Alignment</i> et <i>Translation</i> , un <i>dictionnaire bilingue</i> pour chaque couple de langue, et <i>certain dictionnaires utilisateur</i>) + le format XML (XML-Dict)
LMT, PT	Basé sur la syntaxe de Prolog
MU	Basé sur CommonLisp de Kyodai (Université Kyoto)
METAL	Basé également sur la syntaxe de Lisp

Cette hétérogénéité est une difficulté majeure pour la construction d'une BDLM universelle pouvant gérer à la fois plusieurs systèmes de TA : on ne peut pas gérer tous les aspects approfondis de la structure de tous les systèmes. Par exemple, voici un exemple de dictionnaire monolingue de SYSTRAN et celui de transfert lexical d'un système écrit dans Ariane-G5 :

##	id	lemma	cat	info	morpho
...					
3004	convalescent	A		+DER=(NCE0+MED+ABS+CT+PROGEN)	A15
3005	convective	A			A15
3006	convenient	A		+DER=(ADVLY0,NCE0+ABS+CON+CT+MS)	A15
3007	conventional	A		+DER=(ADVLY0,TY0+ABS+MS+PROPTY)	A15
3008	conventual	A			A15
3009	convergent	A		+DER=(NCE0+MS+ABS)	A15
...					
Dans ce dictionnaire, les champs sont séparés par le caractère tab. Les champs <i>info</i> et <i>morpho</i> contiennent des codes d'attributs spécifiquement réservés au système de TA de Systran. Pour décoder, il faut connaître la déclaration de ces attributs et leurs valeurs, définies ailleurs.					
Dans le dictionnaire de transfert lexical du système anglais-français écrit sous Ariane mentionné au 3.3.1 :					
'ALL'		==	/	/'TOUT'	,\$INT,\$KADDEICT.
'ALLOF'		==	/	/'TOUT'	,\$INT,\$KADDEICT.
Dans ce dictionnaire, on associe des informations grammaticales et					

⁵⁷ Microstructure : organisation interne d'un article de dictionnaire, en fonction du contenu de chaque article (identificateur, mot-vedette, catégorie lexicale, etc.)

des références lexicales aux UL. Chaque entrée peut contenir des conditions et des procédures. Pour décoder, il faut connaître et comprendre la déclaration de ces procédures.

Au pire, on pourrait considérer que le contenu de chaque entrée est du texte brut, mais on obtiendrait/construirait alors plus un entrepôt de dictionnaires de TA au lieu qu'une BDLM.

Un compromis possible est de sélectionner des champs communs (comme le mot-vedette, la catégorie grammaticale, la définition, les exemples...) et de les unifier (pour tous les systèmes), les autres informations restant spécifiques à chaque système, et simplement stockées telles quelles, et éventuellement cachées si les propriétaires de ce système le souhaitent. Chaque système peut alors utiliser ses informations spécifiques comme il le veut ; il suffit que la BD lexicale offre des API pour les trois types de fonctionnement vus plus haut (extraction de dictionnaire complet, accès en lecture pendant l'exécution de la TA, et possibilité de modification pendant la TA).

7.2.2. Variété des opérations sur l'information lexicale

Peut-être à cause de ces difficultés, on a eu très peu de bases de données lexicales réelles pour des systèmes de TA. Nous pouvons citer BDTAO (B'Vital, 1983-95), la base lexicale en Adabase (Lisp) du projet MU (Kyoto, 1982-87), et quelques solutions partielles chez SYSTRAN et autres fournisseurs de systèmes de TA.

Il est alors important de ne pas chercher à comparer les fonctions spécifiques pour chaque système de TA, mais de considérer les fonctions disponibles pour les développeurs des systèmes de TA en général, quand ils sont des connaissances lexicales stockées sous une base de données ou bien dans des fichiers structurés. Ce sont : la récupération et la gestion, le développement lexical, et le traitement des informations lexicales.

7.2.2.1. Récupération et gestion

En ce qui concerne les dictionnaires de TA, il y a plusieurs fonctionnalités possibles pour aider les travaux des lexicographes comme l'indexage, la vérification de cohérence entre LS et LC, *etc.* Normalement, les développeurs construisent des modules *ad hoc* pour supporter ces tâches dans chaque dictionnaire de chaque système de TA (voir 7.1.1). Le développement de telles fonctionnalités pour un grand nombre de systèmes de TA serait très coûteux, et trop coûteux dans le cadre de la recherche.

7.2.2.2. Développement lexical

Les connaissances lexicales constituent le composant qui est le plus modifié et le plus coûteux dans les systèmes de TA et de TH. Il y a plusieurs efforts comme [Queens et Recker-Hamm, 2005] ou Yakusite.net vers la construction d'environnements contributifs en ligne pour développer les connaissances lexicales. C'est un aspect important d'une BDLM.

7.2.2.3. Programmabilité

Des systèmes de TA comme Ariane-G5 ou LMT (7.1.3.3) proposent des opérations très puissantes sur leurs ressources lexicales, par exemple, créer un nouveau couple de langues (L1-L3) à partir des dictionnaires (L1-L2) et (L2-L3), unifier des dictionnaires, *etc.* Donc, une BDLM universelle pour des systèmes de TA devrait supporter la programmabilité sur son information lexicale.

7.2.3. Problèmes pratiques

En parallèle avec les problèmes théoriques, on s'attend à rencontrer également des difficultés pratiques lorsqu'on construira et mettra en service une BDLM universelle. Certaines viennent de l'hésitation ou de la résistance au partage des données et d'autres de la difficulté de construire des environnements adaptés aux usages humains.

7.2.3.1. Partage et protection des données

Les ressources lexicales de chaque dictionnaire coûtent très cher à développer et à maintenir (SYSTRAN recrute 20 linguistes et lexicographes pour développer et maintenir ses dictionnaires, Fujitsu a investi énormément depuis 2001 pour construire 5M entrées supplémentaires de dictionnaires techniques ATLAS-II).

Partager gratuitement ce « trésor lexical » n'est donc pas évident dans l'industrie. Il est donc important de traiter le problème des IPR⁵⁸ et de permettre le partage partiel, limité à certaines parties de l'information et/ou à des sous-ensembles des entrées définis par leurs propriétaires.

7.2.3.2. Synchronisation

Un problème qui apparaît dès la conception d'une BDLM supportant plusieurs systèmes de TA est la synchronisation de cette base avec les dictionnaires ou bases lexicales existants de ces systèmes.

Cela vient de ce que, dans la réalité, chaque système fonctionne avec ses ressources lexicales avec une communauté d'utilisateurs avec leurs propres outils, et le déplacement de ces ressources vers une base commune doit être réalisé progressivement, en les respectant.

7.2.3.3. Synthèse des problèmes

Les expériences ci-dessus aussi bien que les précédentes dans le projet UNL (2.2.3.3) nous ont convaincu que les développeurs des dictionnaires ont besoin d'une base lexicale multilingue commune pour les systèmes de TA hétérogène. Dans ce cas, où les développeurs sont éloignés, la seule solution est aujourd'hui de mettre cette base de données sur le Web, avec assez d'outils conviviaux, de sorte qu'elle soit vraiment employée pour le travail direct, et pas seulement pour la synchronisation épisodique.

7.3. Vers une solution limitée aux systèmes de TAO à « pivot lexical »

7.3.1. Travaux précédents de construction d'une BDLM pour la TA ou la TH

7.3.1.1. Solution « DGT-86 » : N volumes multilingues (« furcoïdes »)

Dans les dictionnaires de transfert d'un système, si l'on a la traduction de N langues, théoriquement, on peut avoir N(N-1) types de dictionnaires bilingues entre ces langues. Si l'on a une base lexicale universelle pour S systèmes traitant chacun les N langues, on a N(N-1)(S-1) types de liens possibles. La représentation de chaque lien varie pour chaque système. Cela compliquerait énormément le développement et demanderait une représentation complexe des liens.

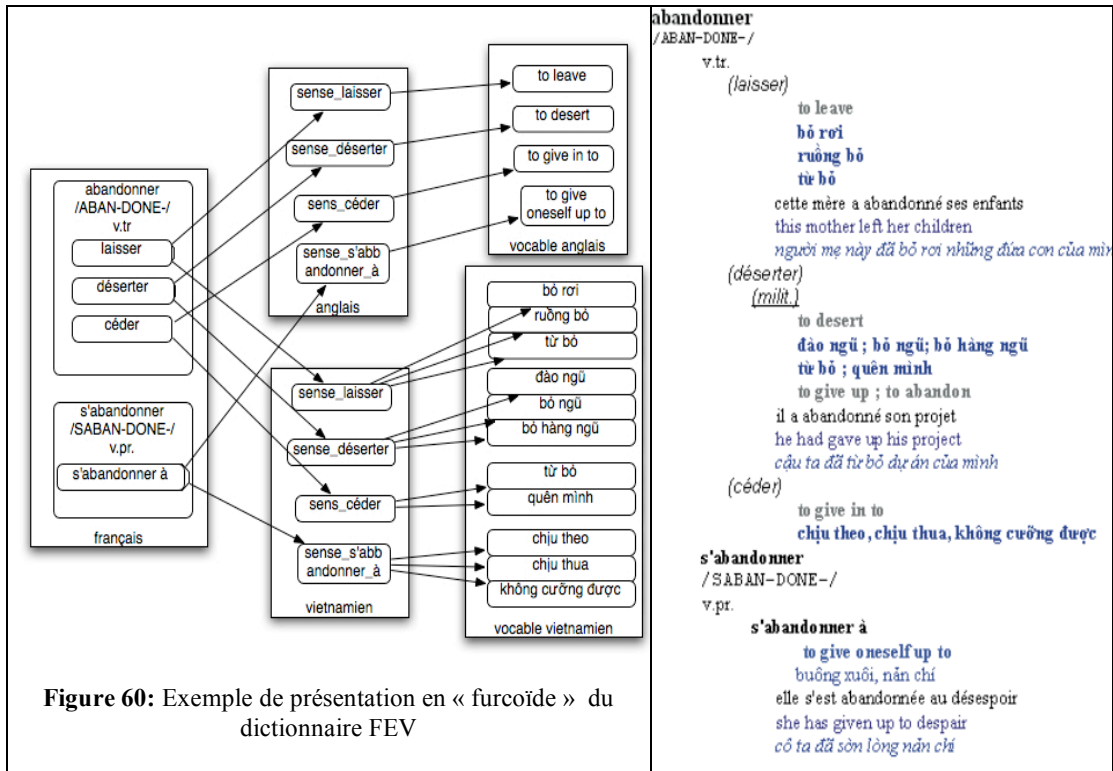
Pour réaliser cela de façon plus simple, on peut séparer le dictionnaire de transfert en 3 parties : 2 monolingues de langue source et cible, et les liens. Cela permet de mettre les parties monolingues dans des volumes où chaque article représente le sens de mot dans cette langue, et les liens multilingues sont groupés entre les langues.

Pour ce faire, on a proposé une solution au sein du projet de construction d'un dictionnaire trilingue français-anglais-japonais (contrat DGT-KDD-GETA-Champollion) : on a utilisé une architecture « en fourche » ou « en éventail » : pour un dictionnaire avec N langues, on présente les entrées dans N volumes monolingues. Chaque entrée représente un sens du mot dans la langue associée. Les correspondances entre Li, Lj sont représentées par des liens. On a une sorte de réseau lexical entre des entrées des N langues. Cette solution a inspiré la construction ultérieure de dictionnaires trilingues (ou multilingues) comme le FEM (français-

⁵⁸ IPR, Intellectual Property Rights, Protection Intellectuelle

anglais-malais) [Gaschler et Lafourcade, 1994], FET (français-anglais-thai) [Lafourcade, 1997], et FEV (français-anglais-vietnamien) [Vo, Phan et Boitet, 2005].

Lors du développement du FEM, on avait commencé avec une macrostructure non pas « furcoïde », mais en séquence : F-E-M. On a vite vu qu'il arrivait souvent qu'un sens français soit traduit en 2 ou 3 sens en anglais, et chaque sens en anglais vers 1 ou 2 sens en malais, créent des duplicata dans le dictionnaire français-malais obtenu en supprimant l'anglais. M. Lafourcade a alors transformé le dictionnaire français-malais en dictionnaire « furcoïde ». Voici un exemple de cette structuration pour le FEV (Figure 60).



Cette solution est pratique et efficace pour les systèmes de TAO monostructure et multicible. Par contre, dans le cas (N → N) fortement multilingue, il faudrait N dictionnaires furcoïdes à N-1 langues cibles, d'où une taille quadratique en N. Pour ce type de situation, il convient de passer par un ou plusieurs « pivots lexicaux » (naturels ou artificiels), ce qui ramène à une taille des dictionnaires linéaire en N.

7.3.1.2. Projet Papillon, une BDLM pour plusieurs dictionnaires

Parmi les efforts de construction de BDLM commune pour plusieurs dictionnaires, on peut citer Papillon-NADIA pour la lexicographie explicative et combinatoire fondée sur des lexies monolingues reliées par des « axes », et le projet Papillon-CDM pour les dictionnaires d'usage général.

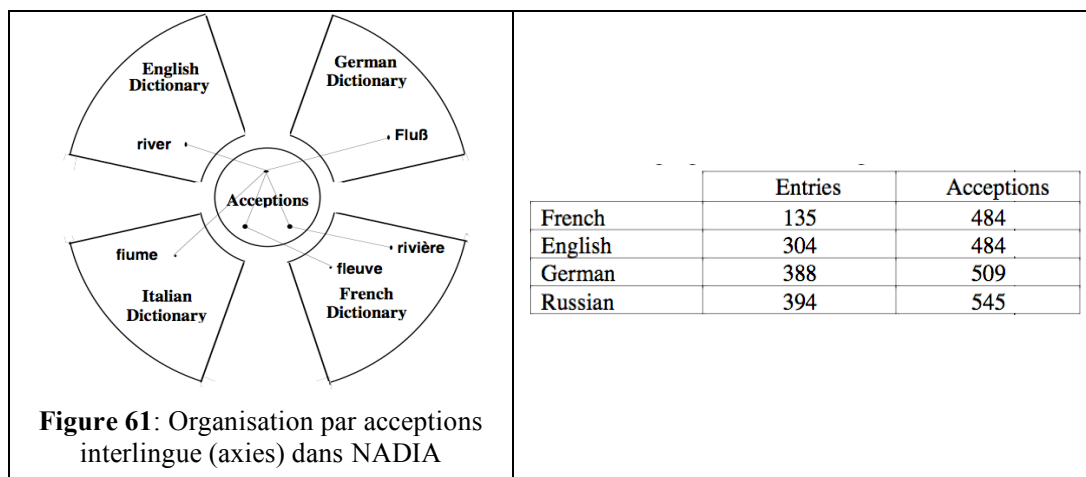
7.3.1.2.1. Papillon NADIA

Le projet NADIA [Sérasset, 1995], lancé au GETA à la suite du projet Multilex ESPRIT, a pour but de construire une BDLM pour les dictionnaires de TALN et de TA. L'expérimentation a été réalisée pour des dictionnaires d'Ariane-G5.

Pour présenter plusieurs dictionnaires, on utilise une macrostructure en étoile. Chaque entrée monolingue est associée à un sens dans sa langue (*lexie*). Le volume interlingue est constitué de liens k-aires (k variable) qui relient les lexies synonymes (en général, plusieurs par langue) et sont appelées « axes » pour « acceptions interlingues » (voir Figure 61). La microstructure d'un volume est définie par un schéma XML avec variantes. Les variantes sont utilisées, entre

autres, pour spécialiser chaque dictionnaire certains attributs et leurs valeurs comme le genre, le cas, le niveau de langue, les dialectes ou régionalisme.

Cette organisation permet d'ajouter une nouvelle langue sans modifier la conception. Cependant, il y a une incohérence si on l'utilise pour gérer les dictionnaires de plusieurs systèmes de TA. Par exemple, si l'on a plusieurs volumes d'une langue naturelle ... comme les volumes français des systèmes écrits sous Ariane-G5, ceux des couples de langues pour SYSTRAN (français-anglais, français-allemand...). il nous faudrait ajouter une couche entre *lexie* et *axie* pour gérer les volumes ayant la même langue.



7.3.1.2.2. Papillon CDM

Le projet Papillon-CDM [Boitet, Mangeot et Serasset, 2002] a été lancé en 2000 pour le but de création d'un environnement de développement lexical collaboratif sur le Web. La motivation initiale du projet est le besoin de dictionnaires français-langues asiatiques pour l'usage humain et pour la machine, en particulier le dictionnaire français-japonais (kanji, kana et romaji) [Tomokiyo, Mangeot et Planas, 2000]. Et puis, cet environnement a été appliqué aux langues thaï, vietnamien, lao, etc. Les données lexicales actuelles sont présentées dans la Table 6.

Table 6: Liste des dictionnaires dans Papillon-CDM

	Nom	Cat.	Type	LS	LC	Entrées	Domaine	Licence
Dict.	WaDokuJiTEn	bilingue	monodirectionnel	jpn	deu	228.801	général	Ulrich Apel
Dict.	ThaiDict	monolingue	monodirectionnel	tha		10.295	général	Kasersart University
Dict.	GDEF	bilingue	direct	est fra	est fra		général	Projet GDEF
Vol.	GDEF_fra			fra		192		
Vol.	GDEF_est			est	fra	68.037		
Dict.	JMdict	bilingue	monodirectionnel	jpn	eng fra deu		général	GNU Public
Vol.	JMdict_jpn_eng			jpn	deu eng fra rus	138.330		
Dict.	FeV	multilingue	monodirectionnel	fra	eng msa tha vie		général	GETA
Vol.	FeV_fra			fra	eng msa tha vie	17.764		
Dict.	FeM	multilingue	monodirectionnel	fra	eng msa		général	GETA
Vol.	FeM_fra			fra	eng msa	19.247		
Dict.	JMdict	bilingue	monodirectionnel	jpn	fra	15 000	général	créé par Jean-Marc Desperrier
Dict.	VDict	bilingue	bidirectionnel	eng fra	vie	58 000	général	créé par Ho Ngoc Duc

Dict.	Néon	bilingue	monodirectionnel	zho	eng	215 424	général	XiaMen
Dict.	FoksEdict	monolingue	monodirectionnel	jpn		4 431 557	général	créé par Slaven Bilac

Pour importer un dictionnaire dans Papillon [Mangeot, 2002], on le transforme en format XML. Si le dictionnaire est bilingue, on sépare les deux parties source et cible en deux volumes. Les liens entre les articles monolingues dans deux volumes sont des entrées d'un volume de liens interlingues.

Voici comment on peut définir une microstructure (structure d'un article monolingue). Nous présentons comme exemple une entrée UW dans le dictionnaire UW d'U++C. Sa structure est définie par le schéma XML suivant.

```
<uw id="unl.upp.abandon.7" status="UN_KNOWN">
  <headword>abandon</headword>
  <pos>VERB</pos>
  <content>abandon(icl>do, agt>thing, obj>thing)</content>
  <definition>stop maintaining or insisting on </definition>
  <examples>
    <example>of ideas or claims</example>
    <example>He abandoned the thought of asking for her hand in
marriage</example>
    <example>Both sides have to give up some claims in these
negotiations</example>
  </examples>
  <more-info>
    <info> </info>
  </more-info>
</uw>
```

À partir de ce schéma, on peut définir la correspondance avec le jeu de balises CDM par le schéma XML :

```
<cdm-elements>
  <cdm-volume xpath="/unl_volume"/>
  <cdm-entry xpath="/unl_volume/uw"/>
  <cdm-entry-id xpath="/unl_volume/uw/@id"/>
  <cdm-headword xpath="/unl_volume/uw/headword/text()"
d:lang="unl" index="true"/>
  <cdm-pos xpath="/unl_volume/uw/pos/text()" d:lang="unl"/>
  <cdm-definition xpath="/unl_volume/uw/content/text()"
d:lang="unl" index="true"/>
  <cdm-example xpath="/unl_volume/uw/examples/example/text()"
d:lang="unl"/>
</cdm-elements>
```

À partir de ce schéma XML, Papillon génère automatiquement le programme d'import et l'interface d'édition adaptée pour ce volume. Par exemple, pour ce volume d'UW, on peut avoir (Figure 62) :

UPP UW Edition Interface

headword: status:

UW:

Definition: POS:

Examples

+	-	<input type="checkbox"/>	<input type="text" value="of ideas or claims"/>
+	-	<input type="checkbox"/>	<input type="text" value="He abandoned the thought of asking for her hand in marriage"/>
+	-	<input type="checkbox"/>	<input type="text" value="Both sides have to give up some claims in these negotiations"/>

Figure 62: Interface d'édition d'une entrée générée par Papillon

Avec cette interface, l'utilisateur peut créer et modifier des entrées. Papillon fournit également des fonctions pratiques comme la gestion d'utilisateurs et de groupes, la recherche simple et avancée, la gestion des versions à la Wiki, l'import/export.

Le projet Papillon nous donne un outil informatique important pour modéliser, gérer et récupérer plusieurs dictionnaires avec des formats et structures variés.

7.3.1.3. Plate-forme Jibiki

La plate-forme Jibiki⁵⁹ [Sérasset, 2004] a été produite en extrayant et en étendant la base logicielle invariante du projet Papillon, pour faciliter la création des BDLM. Elle a été utilisée dans plusieurs projets : GDEF (le Grand Dictionnaire d'Estonien-Français) [Chalvin et Mangeot, 2006], LexAlp (système d'harmonisation de la terminologie juridique sur l'environnement et l'aménagement du territoire dans les Alpes multilingues) [Sérasset, 2006], Papillon (dictionnaires multilingues), MotAMot (Élaboration d'un système lexical multilingue par le biais de la construction de dictionnaires bilingues ciblés sur les langues peu informatisées d'Asie du Sud-Est) [Mangeot et Nguyen, 2009].

La plate-forme Jibiki a donc été testée, validée et utilisée dans plusieurs projets réels, sur plusieurs années. C'est pourquoi nous l'avons utilisée pour développer PIVAX.

7.3.2. Principes d'une solution limitée aux systèmes partageant le pivot lexical

Vu la nature des informations lexicales, notre hypothèse est qu'il est possible de simplifier ce problème et d'arriver à un problème soluble en théorie, implantable, et utile en pratique. Cette simplification a deux aspects.

Le premier consiste à utiliser une macrostructure à « pivot », en étoile, simple à utiliser pour des systèmes de TA partageant un « pivot lexical ».

Le deuxième est qu'on laisse le soin aux développeurs des composants spécifiques à une langue d'un système de TA de gérer eux-mêmes leurs données privées, en particulier leur information codée « en interne ».

7.3.2.1. Macrostructure à avec trois niveaux (lexie, axème, axie)

Pour réaliser une BDLM destinée à plusieurs systèmes de TA utilisant un même pivot lexical, il faut que la structure pivot décrite au 7.3.1.2.1 nous permette de créer, pour chaque dictionnaire un volume monolingue pour chaque langue : par exemple, français du système Ariane-G5, français de Systran, UW provenant d'UNL-C, UW provenant des centres de langues UNL, *etc.* Pour cela, nous avons proposé pour PIVAX une structure (Figure 63) à trois couches : *lexie*, *axème* et *axie*.

⁵⁹ <http://ligforge.imag.fr/projects/jibiki/>

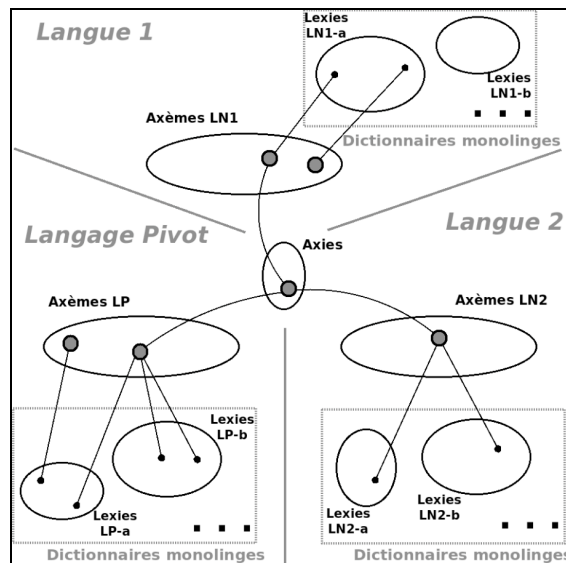


Figure 63: Macrostructure de PIVAX

La macrostructure a 3 niveaux :

- Pour chaque langue naturelle présente ou chaque interlingua (comme l'UNL ou l'IF du projet Nespole!), on a :
 - un ou plusieurs volumes de lexies et leurs informations associées. Une *lexie* correspond à un sens de mot dans un dictionnaire.
 - un unique volume d'axèmes (« acceptions monolingues »). Un *axème* relie des lexies synonymes dans une même langue (i.e., correspondant à une même acception) tandis qu'une acception (monolingue) correspond à un sens « dans la langue ».
- un volume unique d'axies (acceptions interlingues). Une *axie* relie des axèmes correspondant à des sens équivalents (pour la traduction).

Avec cette structure, on peut gérer à la fois plusieurs dictionnaires de TA, comme l'illustre le schéma suivant (Figure 64) :

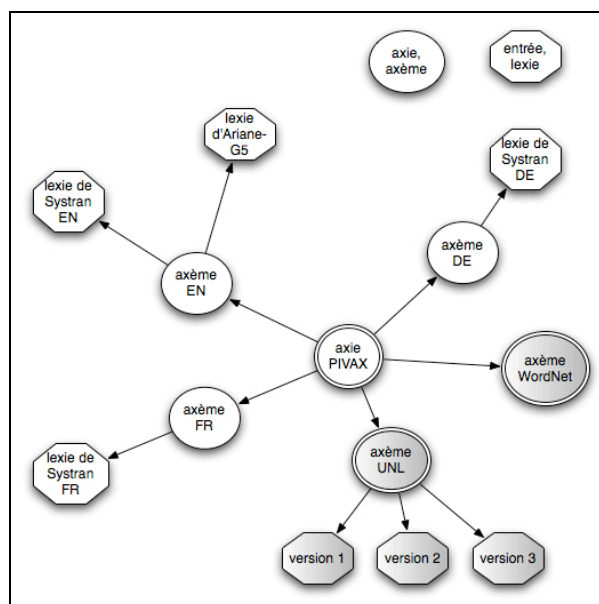


Figure 64: Exemple de volumes dans une base PIVAX

En fait, il y a un point assez subtil à souligner. Notre architecture PIVAX ne met pas le « pivot lexical » au centre du réseau lexical, mais le traite comme un espace lexical d'une langue naturelle. Au centre, on ne met que des axes, classes d'équivalence de (quasi) synonymie (« équivalence traductionnelle »). Cette macrostructure permet une modélisation simple quand la structure linguistique des systèmes de TA est « à pivot », car sinon les axes correspondraient à des liens bilingues.

Une autre raison est que les espaces lexicaux « pivot » ne sont pas assez mûrs pour qu'on puisse les utiliser au centre de la présentation des informations sémantiques. Notre solution permet de relier les lexies à différents ensembles de symboles d'UNL (UW), ou à des symboles d'une ou plusieurs ontologies (ex. IF, WordNet, HowNet...)

Par exemple, UW utilise l'anglais pour les mots-vedettes et des restrictions pour définir le sens de ce mot. En plus, il y a plusieurs ressources d'UW développées par plusieurs partenaires : UNLKB à UNL-C, 280K UW à U++C à partir de Wordnet, UW-LN des centres de langues UNL.

7.3.2.2. Microstructure avec division des données en public/privé

Les **axèmes et axes** sont simplement des liens, qui sont représentés chacun par l'ensemble des identificateurs de lexies ou axèmes correspondants. Comme dans les projets Papillon-NADIA ou LexAlp, on peut relier deux axes ou deux axèmes dans un même volume par un *lien de raffinement*.

Dans les volumes de LN, une entrée contient un lemme, sa classe (POS), son identificateur de sens de mot (le cas échéant), un commentaire (pour les autres développeurs) et le détail approprié de l'information propre à chaque volume et ses codes. Chaque entrée a des méta-données comme la date de modification, l'auteur, l'état de traitement et le niveau de protection, définissant les parties accessibles par le public (droit d'écriture, de lecture et d'exécution).

Comme dit au (7.2.3.1), nous exigeons seulement qu'une information commune minimale soit partagée, celle qui permet d'identifier une lexie (lemme, position, identificateur de sens). Le champ de l'information « propre (propriétaire) » est considéré en tant que donnée textuelle et peut être traité différemment pour chaque volume.

Par exemple, un volume géré par un laboratoire de recherche français pourrait contenir toute l'information française trouvée dans ses systèmes de TA, alors qu'un volume contribué par Systran pourrait contenir seulement une partie de ses codes propriétaires. Voici un exemple :

```
<d:data>
<p:lexie p:id="lexie.systran.test.12004" p:process_status="UNPROCESSED"
p:status="UNKNOWN" p:owner="Systran" p:score="0.00003">
<p:lemma p:access="Public">fier</p:lemma>
<p:class p:access="Public">Adj</p:class>
<p:comment> "fier" can also be a V ("se fier à") </p:comment>
<p:proper_information p:access="Hidden">
/* Systran proprietary codes */
</p:proper_information>
</p:lexie>
</d:data>
```

Pour les volumes interlingues (IL), les entrées sont toujours définies au niveau sémantique, et peuvent donc toujours être appelées « lexies ».

En revanche, les dictionnaires habituels de TA distinguent souvent des sens de mot dans les entrées bilingues de transfert, de sorte que, quand un tel dictionnaire est importé, ses « lexies » doivent plus tard être découpées en lexies réelles (par exemple « fier_Adj » a deux sens et donnera lien à deux lexies).

La microstructure d'une entrée interlingue dépend de l'IL. Dans le cas d'UNL, une entrée contient l'UW comme lexie, un commentaire en anglais au sujet de sa signification, et des exemples d'utilisation.

Avec cette microstructure de PIVAX, l'utilisateur peut définir lui-même les parties d'information de chaque entrée à partager ou à cacher, aux utilisateurs aux groupes d'utilisateurs, et à des projets, grâce à des droits d'accès.

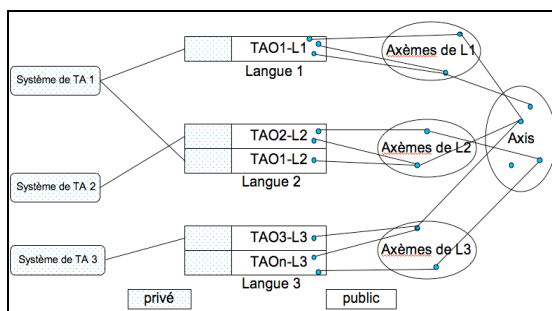


Figure 65: Données publiques et privées en PIVAX

7.3.2.3. Implémentation

Grâce à la plate-forme Jibiki, elle-même implémentée sur Enhydra⁶⁰, nous avons mis seulement 160h pour définir des structures, programmer l'adaptation notre macrostructure à 3 couches et modifier l'interface de navigation à la PARAX et ajouter des fonctions pratiques pour supporter le développement lexical (voir chapitre 8).

7.3.3. Expérimentation sur trois projets

Pour valider l'approche de PIVAX, on l'a expérimentée dans le cadre du projet U++ pour unifier des ressources d'UW en vue de créer un système UNL partageant les UW++ du consortium U++ comme pivot lexical.

En plus, on a prouvé que PIVAX peut aussi supporter des types de dictionnaires bilingues simples comme ceux de SYSTRAN, un système à transfert.

Enfin, on a utilisé PIVAX dans une application de TALN d'annotation de textes dans le cadre du projet ANR, OMNIA.

7.3.3.1. UNL/U++C

Le projet UNL a été lancé en novembre 1996 par l'IAS (UNU, Tokyo). Après 10 ans, on avait une communauté de 16 partenaires et 12 langues avec des outils (EnCo/DeCo, UWGate, ...) et des ressources (spécification des graphes UNL, UNL, des UW, de l'UNLKB, construction de dictionnaires UW-NL sur chaque site ...).

En 2000, une organisation indépendante, UNDL, situé à Genève a été créée pour promouvoir l'application réelle d'UNL. La première application fondée sur UNL a été développée et démontrée en public au 4^{ème} Symposium UNL, à Genève en 2001.

En mai 2004 à Lisbonne, on a proposé avec les centres de langue présents (thaï, coréen, portugais, français, italien, russe et espagnol) l'idée d'un consortium U++C des équipes et personnes qui s'intéressent à l'avancement théorique et pratique d'UNL, avec comme mission principale la dissémination d'UNL.

La fonction de ce consortium est de coordonner les développements de ressources, et d'outils et la mise en service des applications entre des centres de langue. Les projets principaux sont :

⁶⁰ <http://www.enhydra.org/>

- 1) l'unification des ensembles d'UW des différents groupes, pour arriver à un ensemble appelé UW++ et lié à WordNet ;
- 2) la standardisation de la représentation par des graphes UNL, et de son extension (nouveaux attributs, notation des « arguments » linguistiques, représentation des termes composés et des prédicats composés) ;
- 3) la réingénierie des LSPL proposés par UNL-C (bogués, fermés, et non libres de droits) et leur dissémination en licence GPL ;
- 4) la réalisation de nouveaux outils et ressources et leur mise à disposition en utilisation gratuite ou en source ouvert.

Parmi ces tâches, (3) a été faite par l'UPM (EnCo, DeCo), (2) est en cours depuis 2005 (3^{ème} réunion à Paris du 17 au 19 février 2006 et 4^{ème} réunion à Grenoble du 5 au 6 juillet 2007). La partie de « reliage » à WordNet dans (1) a été réalisée par l'UPM [Iraola, 2005]. La partie d'unification des ressources est assurée par PIVAX, et justifie sa construction.

7.3.3.1.1. Le problème d'unification des ressources lexicales (UW)

L'architecture hétérogène d'UNL permet à chaque centre de langue de développer indépendamment son enconvertisseur et son déconvertisseur avec la ressource lexicale UW-LN associée. Bien que l'UNL ait proposé l'outil de développement des UW UWGate, cet outil a été sous-utilisé à cause de ses insuffisances en tant qu'environnement de développement lexical (voir 7.1.2.2).

Ce développement indépendant sur chaque site cause une grande diversité des UW. Par exemple : à Grenoble, on a 20K UNL-français dans deux environnements différents : UNLDeco sur le Web et le déconvertisseur en CASH sous Revolution développé par É. Blanc. À Madrid, on a 200K UW avec l'information de catégorie, la définition, des exemples, construits à partir de PWN (Princeton WordNet), 23K UW sur UNLKB de l'UNL, et des ressources mineures des systèmes expérimentaux UNL-LN.

Par exemple, voici le mot 'access' dans différents dictionnaires d'UW:

U++C
<p>access(icl>recover>do, agt>thing, obj>thing); Cat : VERB; Déf. : obtain or retrieve from a storage device; as of information on a computer;</p> <p>access(icl>reach>do, agt>thing, obj>thing); Cat. : VERB; Déf. : reach or gain access to; Ex. : "How does one access the attic in this house?"; "I cannot get to the T.V. antenna, even if I climb on the roof";</p>
UNLKB
<p>right(icl>abstract thing) access(icl>right) way(icl>abstract thing) abandon(icl>way) access(icl>way) opportunity(icl>time) access(icl>opportunity) market(icl>opportunity) enter(agt>person,obj>thing) access(icl>enter(agt>person,obj>thing)) barge in(agt>person,obj>thing) open(agt>thing,obj>thing) access(icl>open(agt>thing,obj>thing)) reach(agt>thing,obj>thing) access(icl>reach(agt>thing,obj>thing)) use(agt>thing,obj>thing) access(icl>use(agt>thing,obj>thing)) build(icl>use(agt>thing,obj>thing)) drive(icl>use(agt>thing,obj>thing))</p>

UNLDeco/UNL-FR
[accéder] {CAT(CATV),AUX(AVOIR),VAL1(GN),GPI(A)} "access(agt>thing,obj>thing)";
EOLSS/UNL-FR
access(icl>use(agt>thing,obj>thing))

Le consortium U++ (U++C) a donc grand besoin d'une base lexicale universelle pour construire les dictionnaires UW++-LN à partir des dictionnaires UW-LN existants. Cette nécessité a été confirmée dans la 3^{ème} réunion d'U++C à Paris en 2006 [Pérez et al., 2006]. C'était le moment où l'on était en train de concevoir la base lexicale mutualisée PIVAX pour des systèmes de TA hétérogènes. L'application de PIVAX à U++C peut valider ses aspects pratiques.

7.3.3.1.2. Modélisation et montage de la base initiale

7.3.3.1.2.1. Définition des microstructures et de la macrostructure

La microstructure d'un volume UW se compose de : « *mot-vedette* », contenu (restriction), définition, exemples, note et autres informations. Le champ « *note* » est prévu pour être utilisé par des algorithmes divers :

```
<uw id="unl.upp.abandon.4" status="UN_KNOWN">
  <headword>abandon</headword>
  <pos>VERB</pos>
  <content>abandon(icl>do, agt>thing, obj>thing)</content>
  <definition>give up with the intent of never claiming again
  </definition>
  <examples>
    <example>Abandon your life to God</example>
    <example>She gave up her children to her ex-husband when she
    moved to Tahiti</example>
    <example>We gave the drowning victim up for dead</example>
  </examples>
  <note>6,453</note>
  <more-infos>
    <info/
  </more-infos>
</uw>
```

Pour la macrostructure, chaque dictionnaire d'UW est pour l'instant un volume identifié par son origine (système, propriétaire). Pour un dictionnaire bilingue UNL-LN, on crée des axèmes correspondants, puis les axes pour relier les entrées dans les volumes de LN (voir Figure 64).

7.3.3.1.2.2. Gestion des dictionnaires UW-LN

Un dictionnaire UW-LN (de PARAX, UNLDeco...) est décomposé en deux parties : UW et LN avec les informations associées. On importe chaque partie dans un volume de sa langue et on crée les liens aux niveaux d'axème et d'axe. Voici un exemple du mot « ralentir » du dictionnaire de transfert UW-Français du déconvertisseur UNL-FR (écrit en Ariane-G5) présente en format Deco/Enco :

```
[ralentir] {AUX(AVOIR),CAT(CATV),VAL1(GN)} "lower(icl>event,obj>pace)";
[ralentir] {AUX(AVOIR),CAT(CATV)} "decelerate(obj>speed)";
[ralentir]{CAT(CATV),VAL1(GN)} "lower(icl>event,obj>pace)";
[ralentir]{CAT(CATV),VAL1(GN)} "slow_down(icl>event,obj>game)";
```

Après avoir importé dans PIVAX, on peut proposer aux utilisateurs de contribuer [Nguyen, Boitet et Sérasset, 2007] par l'interface (Figure 66). Puis, on peut réexporter les données dans un format prêt à compiler (lemme, attributs, UW) par le compilateur de TRANSF d'Ariane-G5, ou dans un format de dictionnaire Enco/Deco, pour envoi à la base lexicale UWGate/UNLKB de l'UNDL.

FR-UNL transfer dictionary edition interface

headword: status:

Ariane code:

French axeme relation:

Headword	Code
ralentir	AUXIAVOIR CAT CATV
ralentir	CAT CATV VAL IGN
ralentir	CAT CATV VAL IGN

Add a new lexie reference - [click to search target](#)

OR type directly lexie not yet in the lexical base:

headword

code in volume

UW corresponding:

UW	Volume
lower(id>event.obj>pace)	UNLDeco-UNL

Figure 66: entrée 'ralentir' dans l'interface d'édition pour le volume UNL-FR

7.3.3.1.2.3. Import de ressources lexicales externes (TER de G. Ginon)

L'import des ressources en UW a été réalisé durant le stage de TER M1 de G. Ginon [Ginon, 2007]. Il y a deux méthodes de transformation des formats de dictionnaires vers le format XML qu'on peut importer directement dans PIVAX, soit par des script Perl, soit par un outil générique d'accumulation d'ensembles lexicaux structurés à partir de ressources dictionnaires informatisées RÉCUPDIC [Nguyen-Hai, 1998] écrit en MCL (Macintosh Common Lisp).

Avec la collection des UW, on a dans PIVAX (au 1/07/2009) les ressources suivantes:

Ressource	Type/Langue	Nombre des entrées
UNL-Deco	UNL-FRA	39 389
PARAX	UNL-FRA	18 978
PARAX	UNL-CHN (données par Pr. Shi en 2001)	9 315
PARAX	UNL-RUS	13 817
PARAX	UNL-ESP	3 833
UNLKB	UNL	21 618
UPM UW++	UW avec définition et exemples	207 009
UNDL	Projet EOLSS-UNL-FR/UW	21 354
IATE	Projet EOLSS/ENG, termes correspondant aux 21K UW récupérés par Robodico	255 305
IATE	Projet EOLSS/FRA, termes correspondant aux 21K UW récupérés par Robodico	258 175

7.3.3.1.3. Solutions possibles pour unifier les UW

Grâce aux fonctions de Jibiki et à l'introduction d'axèmes dans PIVAX, on a la possibilité d'importer dans PIVAX plusieurs ressources d'UW actuelles comme PARAX, UNLDeco, UNLKB, U++C... Une fois les données rassemblées dans la base PIVAX unique, on peut réaliser des unifications.

Actions élémentaires faisables en manipulation directe. PIVAX est un environnement de développement lexical accessible sur le Web par tous les membres d'U++C. PIVAX permet aux utilisateurs inscrits de contribuer en créant ou modifiant des entrées en mode Wiki. L'utilisateur peut corriger des informations sur les UW, enlever ou rattacher des relations entre des UW et des lexies de LN, trouver et supprimer des duplications...

En se basant sur la vue d'une base PIVAX comme un réseau lexical, [Boitet, 2005] a proposé des primitives pour transformer localement le réseau de façon à maintenir sa cohérence : par exemple, une lexie ne peut pas être connectée à la fois à 2 axèmes ou axes.

Idée d'algorithmes « globaux » impossibles à mettre en œuvre à la main. Vu que l'effort humain est relativement cher et quasiment impossible à réaliser sur un grand ensemble de données (280K d'U++C, 23K d'UNLKB, 30K d'UNLDeco à Grenoble), on cherche une méthode (semi)automatique pour unifier les UW. On souhaite avoir un algorithme « global » sur les ensembles des lexies, axes et axèmes, vus comme des graphes lexicaux. L'étude des algorithmes sur graphes faite par Vincent Archer dans sa thèse (bibliothèque MuLLing, [Archer, 2009]) pourrait être directement appliquée.

Idée d'utilisation d'un langage « narratif ». Une autre grande possibilité pour raffiner ou enrichir des ensemble d'UW est de proposer un LSPL pour que les utilisateurs puissent définir eux-mêmes des contraintes de correspondance des UW entre des volumes, des opérations (supprimer, ajouter, remplacer, séparer ... des lexies, axes ou axes) et la présentation de la sortie souhaitée.

Vecteurs conceptuels. Enfin, il suffirait de modifier légèrement PIVAX pour pouvoir appliquer des techniques de vecteurs conceptuels [Lafourcade, Prince et Schwab, 2002].

7.3.3.2. Gestion de dictionnaires de SYSTRAN

Rappelons que PIVAX a été initialement conçu pour des systèmes de TA partageant un même pivot lexical. Cependant, sa structure est assez générique, et on a pu supporter directement des dictionnaires bilingues de systèmes de type transfert comme SYSTRAN.

Table 7: Exemple de dictionnaire de transfert d'ENAR (anglais-arabe) de SYSTRAN 6

3048	accord with	verb	تجاوب	verb:plain
3050	accordance	noun	مُطَابَقَة	noun:common
3051	accordant	adj	مُلَائِم	
3052	according as	conj	ما وفق	conj:plain

Pour ce dictionnaire, on sépare l'information en parties : entrées monolingues anglais et arabe, axes anglais et arabe, et une axie par entrée dans le dictionnaire de transfert original de SYSTRAN.

Maintenant, on peut créer un prototype de dictionnaire transfert français-arabe pour le système SYSTRAN via le dictionnaire UW-français dans U++C. On compare les mots dans le volume de lexies de l'anglais de SYSTRAN avec les mot-vedettes des UW dans le dictionnaire UW-français, puis on groupe ces deux correspondances dans un nouveau dictionnaire. C'est le principe de la création d'un nouveau couple de langues (L1-L3) à partir des dictionnaires (L1-L2) et (L2-L3) étudié dans la thèse de Nguyen Hai-Doan [Nguyen-Hai, 1998]. Ce dictionnaire français-arabe sera proposé aux lexicographes de SYSTRAN à corriger.

7.3.3.3. Génération de dictionnaires spécifiques dans le projet OMNIA

Une autre expérimentation de PIVAX a été de créer un dictionnaire permettant l'annotation d'un texte « compagnon » d'une image par des UW dans le cadre du projet ANR OMNIA [Rouquet et Nguyen, 2009]. L'originalité est que (1) les dictionnaires d'annotation ainsi construits sont chacun spécifiques à un (petit) texte (il y a 500 000 textes de 60 mots en moyenne); (2) chaque dictionnaire ainsi « extrait » de PIVAX se présente comme un ensemble de règles de réécriture sur des graphes de chaînes d'arbres simplement étiquetés (dans le formalisme des Systèmes-Q); (3) tant l'extraction d'un tel minidictionnaire spécialisé à un texte que son exécution sur ce texte (produisant le texte annoté par des UW sous forme d'un graphe-Q, voir 4.2.2.3.2) peuvent se faire à la volée, ce qui permet de traiter de la même façon l'annotation des textes existants (en principe faite d'avance, en arrière-plan) et celle d'un nouveau texte (à ajouter à la base de données et à la A-box de l'ontologie

OMNIA) ou d'une requête (de quelques mots à un page). Le corpus utilisé est Belga-news, de la campagne CLEF-09.

Dans cette application, PIVAX fonctionne comme un serveur lexical qui produit un ensemble « d'articles » des UW sous la forme spécifique de règles-Q, pour chaque mot envoyé par le module d'annotation d'OMNIA.

Conclusion

Dans ce chapitre, on a vu la construction d'une première BDLM, PIVAX, capable de gérer les connaissances lexicales de plusieurs systèmes de TA partageant un même pivot lexical, ce qui est essentiel pour construire un système de TA hétérogène.

PIVAX est le premier système contributif de base de données lexicales en ligne qui permet de créer, maintenir et gérer les ressources lexicales de systèmes de TA basés sur un « pivot lexical », et hétérogènes dans le sens où leurs composants spécifiques à une langue peuvent être développées à différents endroits et avec différentes approches linguistiques et différents outils informatiques. Seule l'information lexicale spécifique à une langue et essentielle pour la traduction doit être stockée dans PIVAX, de sorte que les développeurs peuvent utiliser leurs propres outils et protéger les informations propriétaires.

PIVAX est effectivement utilisé dans le cadre de plusieurs projets. Dans le projet U++C, il s'agit d'unifier les ressources lexicales pour le système hétérogène UNL. Dans le projet EOLSS/Unesco-FR et dans le projet iMAG, PIVAX est utilisé comme serveur lexical pour les données terminologiques récupérées sur le Web, et pour la base « préterminologique » construite à partir des traductions et des contributions lexicales directes des post-éditeurs.

Dans le projet OMNIA, PIVAX est utilisé pour fabriquer à la volée des dictionnaires langue-UNL sous forme de « systèmes-Q ». Cette application démontre la possibilité d'utiliser PIVAX comme serveur lexical en TA, puisque les dictionnaires produits pourraient aussi bien être des dictionnaires de TA (ici, d'enconversion ou de déconversion).

Chapitre 8. Évolution de PIVAX de la TA hétérogène à la TAO hétérogène

Introduction

Nous avons partiellement résolu le problème du support des systèmes de TA hétérogènes par la construction de PIVAX et sa validation sur 3 projets. Il nous reste à l'étendre pour permettre le développement lexical contributif sur un même environnement.

Le support au développement lexical par des humains est un aspect connu. On a vu des efforts pour faciliter ce processus par un environnement adéquat, et en particulier en collaboration comme [Queens et Recker-Hamm, 2005].

Cependant, un support pour à la fois les dictionnaires de TA et l'usage général dans la traduction humaine en vue d'unifier les deux types de ressource n'existe pas beaucoup, à part Yakushite.net [Murata et al., 2003] et le projet DGT-86 [Boitet et Nedobejkine, 1986a].

À partir de ces besoins, nous avons étendu PIVAX de 3 façons en en faisant une BDLM contributive, en le rendant utilisable directement en THAM, et enfin en l'intégrant des des projets réels de traduction.

L'aspect le plus important est d'offrir une base lexicale contributive, qui puisse servir à des utilisateurs différents (les lexicographes ou linguistes dans les systèmes de TA, les traducteurs pour gérer les dictionnaires dans les projets de TH, les contributeurs bénévoles, les utilisateurs y accédant pour consulter à la volée...). PIVAX doit fournir les fonctions générales pour supporter ces différents usages sous un même environnement.

Ensuite, pour mieux utiliser PIVAX pour la TH, il faut qu'on l'intègre dans les systèmes de THAM, comme SECTra_w, utilisé pour la traduction et la post-édition, pour gérer la base lexicale de ces systèmes. Cette intégration permettra de valider PIVAX dans des projets réels de traduction et d'augmenter son utilisation ainsi que les ressources.

L'intégration de PIVAX dans des systèmes différents s'unifie ainsi avec la construction de systèmes de TAO hétérogènes.

Dans ce chapitre, nous présentons d'abord dans la section 1 la transformation de PIVAX en une base lexicale contributive. Puis, dans la section 2, on étudiera et expérimentera certains nouveaux composants nécessaires pour accélérer l'utilisabilité directe de son environnement. Dans la section 3, on présentera l'intégration de PIVAX dans les systèmes de THAM pour gérer leur base lexicale dans des projets réels de traduction.

8.1. PIVAX comme une BDLM contributive

Les connaissances lexicales sont le composant le plus coûteux et le plus modifié parmi les logiciels des systèmes de TA et de THAM. Il y a eu beaucoup d'efforts pour construire des environnements de développement lexical collaboratifs comme [Queens et Recker-Hamm, 2005] ou les projets réalisés à partir de la plate-forme Jibiki : GDEF, LexALP, Papillon.

Pour PIVAX, on cherche alors à réaliser un environnement adéquat pour supporter le mieux possible le travail des lexicographes. Les fonctions de PIVAX par rapport à cet aspect ont été définies à partir d'une synthèse des expériences sur des EDL précédents pour le développement lexical.

8.1.1. Expériences sur des environnements de développement lexical

Une fonction majeure d'une BDL est la gestion et la contribution des ressources par des humains (lexicographes, bénévoles, utilisateurs naïfs, ...). Pour cela, il faut proposer un environnement associé à chaque BDL. L'étude de quelques environnements de

développement lexical permet de dégager un certain nombre de points essentiels, à prendre en compte sous peine de rejet par les utilisateurs.

- **MMT-CICC.** Le projet MMT (Multilingual MT) de CICC (voir 2.2.3.2) a un environnement centralisé, mais non accessible à distance. La communication est réalisée par les fichiers de formulaires remplis dans différents endroits. Ce n'était alors pas un vrai EDL efficace pour le développement lexical.
- **UNLDeco.** C'est un service Web développé par G. Sérasset [Sérasset et Blanc, 2003] pour déconvertir des graphes UNL en français (voir le projet UNL, 2.2.3.3). Il y a une base de données lexicales accessible sur le Web en temps réel, mais limité au dictionnaire de transfert UNL-français du système UNL-FRA écrit sous Ariane.

On aurait voulu l'utiliser pour développer le dictionnaire UNL-FRA, et les systèmes UNL-russe, UNL-espagnol dans le cadre du projet de traduction du site B@bel de l'Unesco [Boitet, Boguslavskij et Cardeñosa, 2007]. On ne l'a finalement pas fait. La raison est qu'on n'avait pas de support pour le travail lexicographique comme le tri, le filtrage ou l'assistance à l'indexage (introduction des codes linguistiques) dans les dictionnaires. On ne garde pas les anciennes versions de modification faite par les utilisateurs. On n'avait pas non plus d'interface utilisateur aussi conviviale et puissante que celle de PARAX (voir 7.1.2.3).

- **UWGate.** pour accéder au dictionnaire des UW et UNLKB par UWGate d'UNL-C (voir 7.1.2.2), il faut qu'on installe en local l'interface graphique donnée par UNDL. L'échange entre cette interface et la base centrale est réalisé par échange de fichiers, même si l'on a une entrée dans le dictionnaire. En plus, le temps d'attente est souvent très long (2 ou 3 jours ou jamais).
- **PARAX.** L'interface de PARAX (voir 7.1.2.3) a été conçue pour le développement lexical. On a un champ pour saisir le mot vedette, la liste des mots trouvés avec le sens est affichée dans la colonne à gauche. Quand l'utilisateur clique sur un UW, les correspondances dans les langues associées (japonais, russe, français) sont affichées dans les colonnes de droite.

C'est une interface pratique qu'un linguiste peut utiliser pendant longtemps. On pourra s'en inspirer dans notre nouvel environnement pour que la BDLM puisse être utilisée vraiment à la fois pour la TA et la THAM.

8.1.2. Scénario

Dans le contexte de la création d'une BDLM contributive, on veut que PIVAX soit capable de supporter les scénarios typiques suivants.

Contributeur. Un contributeur ou un linguiste peut se connecter et contribue aux lexies du pivot interlingue (ex. UW) et des langues naturelles, selon ses droits d'accès. Chaque utilisateur a un profil et un espace de travail. Si deux utilisateurs travaillent sur une même entrée, PIVAX le leur signale. On gère les versions de contribution sur chaque entrée comme dans Wikipedia. On a toujours la possibilité de revenir à une version précédente si nécessaire.

Administrateur. PIVAX propose les fonctions de gestion et d'administration des dictionnaires comme l'import, l'export, la gestion de groupes et d'utilisateurs.

Modérateur (gestion de ressources). Ce sont des professionnels ou des chefs de projet qui gèrent un projet de développement lexical. Ils affectent des entrées à des utilisateurs, et révisent leurs contributions.

Développeur informatique. Il peut corriger des bogues, utiliser le PIVAX par API.

Développeur linguistique. Pour mieux utiliser PIVAX dans les applications, on envisage d'y ajouter de la programmabilité (voir chapitre 9) : l'utilisateur ayant certaines compétences a

priori faible, pourrait définir lui-même des tâches qu'il veut, comme l'unification de certaines ressources.

8.1.3. Spécification externe

8.1.3.1. Interface de navigation à la PARAX

Nous avons développé une interface à la PARAX, avec les entrées présentées dans des colonnes, une par langue (Figure 67). Chaque langue d'un système est mise dans une colonne. On peut changer l'ordre des colonnes ou leurs largeurs en cliquant sur les boutons (-xxx+). Les paramètres de présentation des colonnes sont mémorisés dans le profil d'utilisateur. Les entrées reliées à une même axie ou à un même axème sont présentées en leur affectant une même couleur.

Le résultat des recherches est mis en page. Lors de la navigation dans un résultat de recherche, pour chaque entrée, on peut demander de l'éditer, de la dupliquer, d'afficher son format XML, ou de l'ajouter dans un panier pour y revenir ultérieurement.

Les entrées en fond grisé sont en cours d'en édition par un autre utilisateur.

The screenshot shows a search result page for the word 'tester'. The page is organized into a grid of columns representing different languages and systems. The left sidebar contains user information (User: nguyent, Language: english) and navigation options (User Profile, Sign out, Languages, Lookup, Advanced Lookup, Dictionary List, Entries). The main content area is titled 'Search result' and shows '4 entry(ies) retrieved.' with a 'Next entries' link. The grid displays entries for 'ariane' with columns for '- fra +', '< - fra.systran + >', and '< - fra.axeme + >'. Each entry includes the word 'tester', its volume and profile information (e.g., '(V, Prl, 2) ariane.fra.testers.3'), and action buttons (EDIT, DUPLICATE, DELETE, HISTORY, MORE +). Some entries are highlighted in grey, indicating they are being edited by another user.

Figure 67: Interface de navigation d'un résultat de recherche

8.1.3.2. Interface d'édition d'une entrée

Pour l'édition des entrées dans un volume, on dispose d'une interface créée pour ce volume. Par exemple, quand on clique sur (EDIT) de l'entrée « tester », on arrive à l'interface présentée dans la (Figure 68) et la (Figure 69).

Nous avons utilisé la possibilité offerte par Jibiki de « spécialiser » l'interface automatiquement produite par Jibiki à partir de la définition de la microstructure de chaque volume.

Quand on veut attacher une correspondance monolingue (axème) ou multilingue (axie), on clique sur « click to search target », une fenêtre pop-up est ouverte, et on y recherche les entrées dans les autres langues des dictionnaires des systèmes.

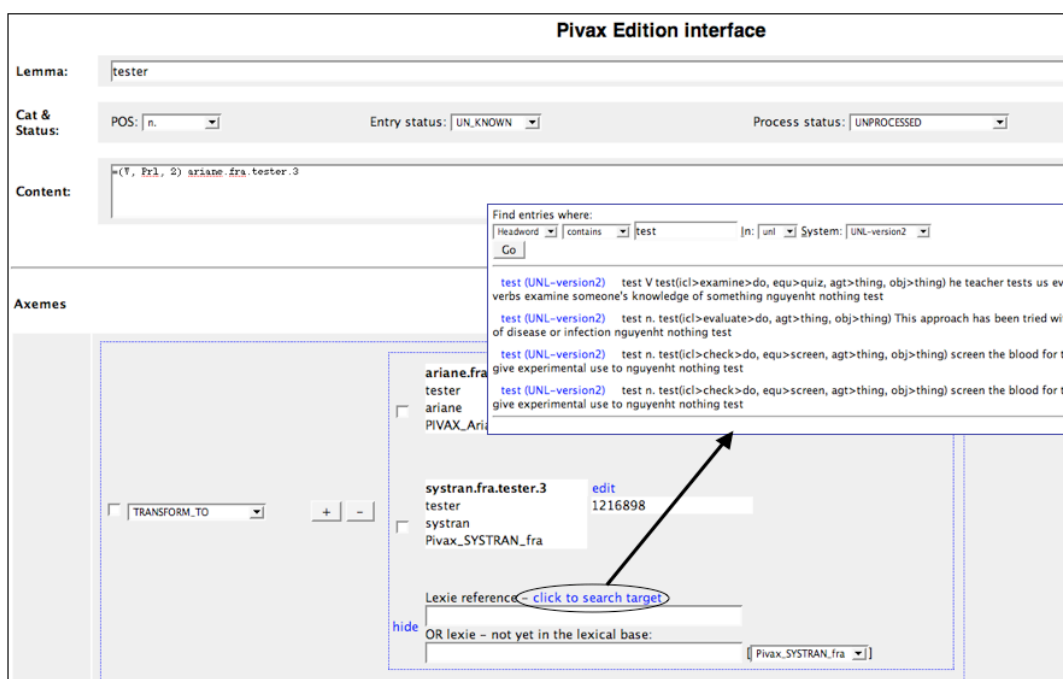


Figure 68: Interface de l'édition d'une entrée (1)

On peut modifier l'information et cliquer sur (Update) pour sauvegarder, mais on continue l'édition et les autres utilisateurs ne peuvent pas éditer cette entrée. Quand on finit l'édition, on clique sur (Finish), cette entrée est déverrouillée, et d'autres peuvent l'éditer.



Figure 69: Interface de l'édition d'une entrée (2)

8.1.3.3. Recherche simple et avancée

Pour la recherche avancée, grâce à Jibiki, on peut combiner certains critères de recherche. Par exemple, ne considérer que les entrées des volumes d'un projet, et se limiter aux entrées ayant un statut particulier (en cours d'édition), *etc.* Ces critères doivent pouvoir être mémorisés sous un alias pour qu'on les reprenne à la prochaine session.

The screenshot shows the 'Advanced search interface' with the following configuration:

- Find entries in:** Pivax
- where:**
 - Headword is test in all languages
 - POS contains M in all languages
- Systems:** UNLKB, systran, UNL-version2, UNL-version1, ariane, UPM, UNLDecofRSite, PARAX
- Show target languages:** English, Spanish, French, Russian, UNL
- Display:** 10 results per page using form: Default
- Remember search as:** test with POS like 'M'
- Shortcut(s):** test saving | tester_u++demo

Figure 70: Interface de composition des critères de recherche avancée

8.1.3.4. Profil d'utilisateur

Avec les informations personnelles d'utilisateur (mot de passe, groupe, ...), le profil contient également des paramètres mémorisés lors de l'utilisation : présentation des colonnes (ordre, largeur, langues à afficher), liste des articles choisis lors de la navigation (panier), noms des critères de recherche avancée.

Chaque utilisateur peut avoir plusieurs profils, mais un seul est actif lors d'une session. On peut aussi affecter un profil à un groupe ou au public tout entier (Figure 71).

Hong-Thai NGUYEN Profile					
Quick Search Alias					
Existing Quick Search Alias	Owner	Remove	Share type	Share group/user	
test saving	Hong-Thai NGUYEN	remove	public	<input type="text"/>	
tester_u++demo	Hong-Thai NGUYEN	remove	public	<input type="text"/>	
<input type="button" value="Update"/>					
User Profile					
Profile name	Owner	Remove	Status	Share type	Share group/user
test	Hong-Thai NGUYEN	remove	INACTIVE	public	<input type="text"/>
test1	Hong-Thai NGUYEN	remove	ACTIVE	public	<input type="text"/>
test2	Hong-Thai NGUYEN	remove	INACTIVE	public	<input type="text"/>
<input type="button" value="ProfileUpdate"/>					

Figure 71: Plusieurs profils possibles pour un utilisateur

User Profile

Profile name	Owner	Remove	Status	Share type	Share group/user
test	Hong-Thai NGUYEN	remove	INACTIVE	public	
test1	Hong-Thai NGUYEN	remove	ACTIVE	public	
test2	Hong-Thai NGUYEN	remove	INACTIVE	public	

ProfileUpdate

Profile detail

Name:

Type:

Description:

Adjust:

Language	Column size	Show	Remove	Order
<input type="text"/>	<input type="text" value="20"/>	<input checked="" type="checkbox"/>		
unl	<input type="text" value="200"/>	<input checked="" type="checkbox"/>	remove	up down
fra	<input type="text" value="300"/>	<input checked="" type="checkbox"/>	remove	up down
eng	<input type="text" value="400"/>	<input checked="" type="checkbox"/>	remove	up down

Update Reset Create

Figure 72: Un profil avec les paramètres de présentation des colonnes

Pour développer PIVAX, on a utilisé la plate-forme Jibiki (voir 7.3.1.3) version 1.0 implémentée sous l'environnement Enhydra avec le SGBD PostgreSQL. Nous avons pris environ 2 semaines pour apprendre comment décrire la macrostructure et les microstructures pour tous les volumes des dictionnaires UW et langues associés. Il nous a fallu environ 90 heures de programmation pour adapter Jibiki à notre nouvelle macrostructure : gestion des volumes d'axèmes, introduction de nouvelles fonctions pratiques (navigation en colonnes, profil d'utilisateur, panier, alias dans la recherche avancée...).

8.2. Nouveaux composants pour l'utilisation directe en THAM

Pour augmenter l'utilisabilité de l'environnement de PIVAX, on peut y intégrer de nouveaux composants. Il s'agit de (1) la recherche améliorée avec ou non la lemmatisation pour mieux exploiter le résultat de recherche dans la base et (2) des aides pour supporter le travail lexicographique comme l'indexage, la manipulation directe des éléments dans la base, et la gestion de flux de travail. (3) Pour la ressource, on cherche à remplir automatiquement le dictionnaire à développer par les ressources extérieures disponibles.

8.2.1. Recherche exacte et approchée intégrant la lemmatisation

Les méthodes de recherche exacte et approchée présentées au 6.1.1.1 peuvent être expérimentées pour mieux exploiter la base.

Une autre forme de recherche approchée consiste à passer les mots à chercher au module de lemmatisation, comme INTEX/NOOJ⁶¹ de Max Silberstein. Le résultat de lemmatisation est multiple (ex. : avions, car, ferme...), et chaque lemme trouvé est recherché dans la base.

Les mots composés sont aussi pris en compte. Par exemple, la suite de mots: « ... white wall paper ... » peut être segmentée en 5 mots simples ou composés : white wall, wall paper, white paper wall, white, wall, paper. On utilise dans le projet OMNIA (voir 4.2.2.3.2) un format compact d'échange de résultats multiples de lemmatisation, à savoir un graphe-Q :

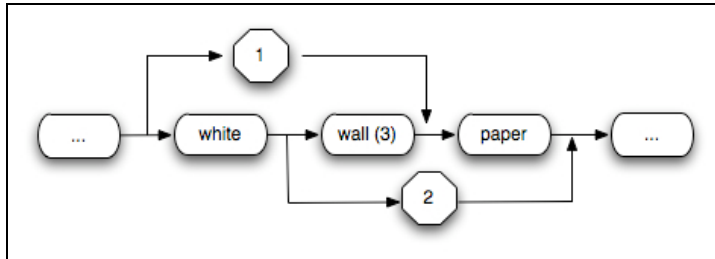


Figure 73: Segmentation multiple des occurrences

8.2.2. Intégration des aides de développement lexical avancées

Parmi les fonctions d'aide au travail lexicographique direct sous PIVAX, on insiste sur (1) l'indexage, qui a été réalisé par ATLAS au GETA, (2) la manipulation directe du réseau lexical en tant que graphe qui est facile à réaliser, grâce à la conception de réseau lexical implicite dans l'organisation de PIVAX, et (3) l'intégration des outils de gestion de flots de travaux.

8.2.2.1. Indexage (ATLAS)

Pour supporter le développement de dictionnaires de TA dans PIVAX, il nous faut indexer la partie « code propriétaire » dans le volume monolingue de chaque système. Pour cela, nous avons récupéré l'outil d'indexage ATLAS (voir 3.2.3.3.1) écrit en Pascal par D. Bachut pour écrire et interpréter des « cartes d'indexage ».

8.2.2.2. Outil intuitif de manipulation de graphe

Dès le début de la conception de PIVAX, on a considéré les données lexicales comme un réseau lexical. Le développement lexical peut alors être fait comme une manipulation directe de graphe par une interface plus intuitive que les formulaires actuels.

Pour réaliser cela, on suit le principe décrit dans 4.2.1.2.1 de l'unicité d'une structure interne généralisant la structure abstraite dans l'implémentation. On transforme les données de PIVAX en une structure interne de graphe et on utilise un outil de présentation et de manipulation graphique (comme GraphViz). La manipulation de graphe sous l'interface graphique est donc déléguée à GraphViz, et les changements sont répercutés dans la base de données PIVAX.

8.2.2.3. Intégration des outils de gestion de flot de travaux

Le travail sur les flots de travaux pour le développement lexical a été commencé par le M2R de Rami Albatal [Albatal, 2005] avec G. Sérasset en 2006. Cependant, ce travail s'est limité à sélectionner des outils adéquats de gestion de flots de travaux à intégrer. Il nous faut une étude plus approfondie sur les tâches et données spécifiques au développement lexicographique en liaison avec la THAM afin de pouvoir concevoir et implémenter une plate-forme intégrée de gestion de flots de travaux spécifique supportant l'extraction des

⁶¹ <http://www.nooj4nlp.net>

termes à partir des corpus, le développement collaboratif, la révision, la validation,... Certaines de ces tâches ont été introduites par M. Dillinger dans LOGOS [Dillinger, 2001] et [Tesconi et al., 2006].

8.2.3. Préremplissage de données lexicales par des agents externes

Pour réduire le coût de développement des dictionnaires, on cherche d'abord à les remplir automatiquement, puis on propose aux lexicographes de les corriger. Ce travail peut être divisé en trois approches selon l'origine des ressources à exploiter : (1) dictionnaires disponibles dans un format électronique connu, (2) ressources disponibles sur le Web comme IATE⁶², (3) termes bilingues à partir de corpus spécifiques.

Le travail (1) de récupération de dictionnaires sur un format en local a été mené par [Nguyen-Hai, 1998]. On a appliqué cette méthode pour plusieurs dictionnaires de Papillon [Teeraparbserree, 2002].

Une méthode de récupération (2) de dictionnaires en ligne a proposée par M. Mangeot en 2001. Il s'agit d'un moteur de recherche automatique des correspondances dans les langues cibles pour les mots à chercher dans une langue source. Dans le cadre du projet EOLSS/UNL-FR, nous avons repris cette idée et implémenté Robodico pour fouiller des dictionnaires en ligne comme IATE, Wiktionary...

L'extraction des termes bilingues d'un corpus (3) est actuellement fournie par plusieurs outils qu'on peut les intégrer dans PIVAX comme des agents externes (ex. Robodico).

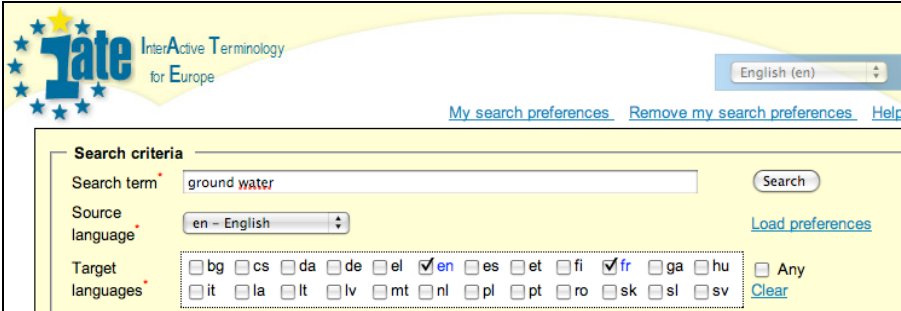
8.2.3.1. Robodico

À partir des mots d'un projet de traduction comme EOLSS/UNL-FR (220K mots), on lance Robodico sur un site de dictionnaires ou de terminologie comme IATE. Le mécanisme de Robodico consiste à simuler les paramètres demandés dans la requête HTTP sur l'interface du site (Figure 74). On envoie la requête pour chaque mot et on récupère la page HTML contenant le résultat de la requête. On extrait les informations pertinentes et on les sauvegarde sous un format adéquat en local.

Dans l'exemple suivant, tiré du projet EOLSS/UNL-FR, on ne demande que la traduction d'anglais en français pour les 15K mots ou termes apparaissant comme « mots-vedettes » dans les 136 76 graphes UNL correspondant aux segments des 25 articles de EOLSS préparés par l'UNDL (220K mots en total).

La requête http équivalente est:

<http://iate.europa.eu/iatediff/SearchByQueryLoad.do?query=ground%20water&sourceLanguage=en&targeten=true&targetfr=true>



The screenshot shows the IATE search interface. At the top left is the IATE logo with the text 'InterActive Terminology for Europe'. On the right, there is a dropdown menu for 'English (en)'. Below this are links for 'My search preferences', 'Remove my search preferences', and 'Help'. The main search area is titled 'Search criteria' and contains a search term input field with 'ground water', a 'Search' button, a source language dropdown set to 'en - English', and a 'Load preferences' link. Under 'Target languages', there is a grid of checkboxes for various languages, with 'en' and 'fr' checked. Other languages listed include bg, cs, da, de, el, es, et, fi, ga, hu, it, la, lt, lv, mt, nl, pl, pt, ro, sk, sl, sv, and 'Any'.

Figure 74: Interface d'envoi d'une requête d'IATE

⁶² <http://iate.europa.eu/iatediff/SearchByQueryLoad.do>

On récupère une page HTML avec des informations intéressantes : mot-vedette (exact et approché), domaine, note (sous forme d'icônes en étoile) et la référence du terme dans la base IATE (Figure 75). On extrait ces informations et on les met dans un fichier local.

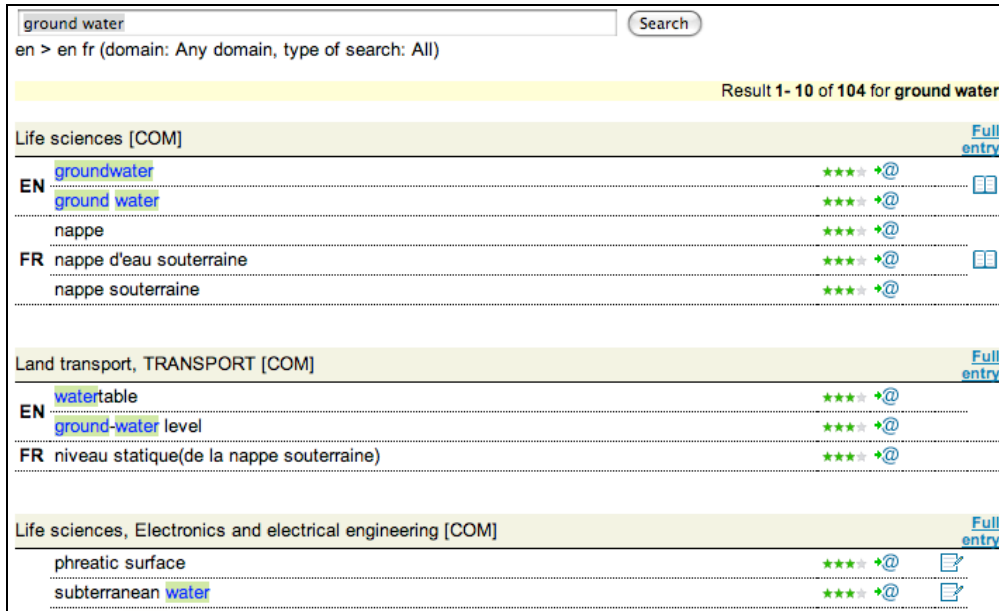


Figure 75: Résultat renvoyé par IATE pour le mot (terme composé) « ground water »

Nous avons appliqué ici la même technique que celle inventée par M. Mangeot pour l'import de dictionnaires dans Papillon-CDM (voir 7.3.1.2.2).

ID de mot envoyé	Langue	Correspondance	Domaine	Score
...				
EOLSS.unl.ground water.1	en	ground%20water ground%20water	Life sciences[COM]3	
EOLSS.unl.ground water.1	en	ground%20water ground%20water	Life sciences[COM]3	
EOLSS.unl.ground water.1	fr	nappe	Life sciences[COM]3	
EOLSS.unl.ground water.1	fr	nappe d'eau souterraine	Life sciences[COM]3	
EOLSS.unl.ground water.1	fr	nappe souterraine	Life sciences[COM]3	
...				

On importe ces ressources dans PIVAX et on les rend accessible par une interface de consultation et d'édition.

8.2.3.2. Outils d'extraction de termes bilingues à partir de corpus

Les méthodes d'extraction de termes et de dictionnaires à partir de corpus ont été étudiées par plusieurs travaux :

- Documents à traduire, MT : module d'extraction des termes des systèmes commerciaux de gestion de MT ou des bureaux de traduction comme :
 - Multilingue : Trados (MultiTerm), TermServer [TermServer, 2009],
 - Monolingue : outils d'extraction des termes de Translated-Lab.net [Translated-Lab, 2009], de TrM Translation [TrM-Translation, 2009], d'Orchestre [Orchestre8, 2009], du format TMX [Heartsome, 2009], ...
- Corpus pour un domaine ou un sous-langage spécialisé : LexTract [Lebarbé, 2007] pour l'extraction des termes à portée juridique (utilisé dans le projet LexAlp), outil d'extraction des termes pour médecine fondé sur une analyse morphologique ([Bernhard, 2006])
- Sur le Web : des corpus sur Internet [Gaiffe, Jacquey et Kister, 2009].

Tous ces outils peuvent être utilisés en externe pour préremplir la base initiale de PIVAX à partir de corpus bilingues spécifiques.

8.3. Intégration de PIVAX dans des systèmes de THAM

Bien que PIVAX offre toutes les fonctions nécessaires pour la contribution lexicale, son usage effectif est encore limité. La raison est que chaque centre de langue a déjà son propre outil de développement et que le changement d'habitude des utilisateurs doit être fait progressivement.

Une des pistes pour augmenter son utilisabilité est de l'intégrer dans les systèmes de THAM utilisant une base lexicale liée à une MT, comme SECTra_w (système d'exploitation de corpus et de post-édition) ou iMAG (une passerelle d'accès multilingue contributif à des sites Web), et les projets réels de traduction comme EOLSS/UNL-FR.

Pour cela il faut ajouter des fonctions supplémentaires permettant certains supports dans ces systèmes. PIVAX se dirige vers les systèmes de TAO hétérogènes, et fonctionne comme un serveur lexical dans ce contexte-là.

8.3.1. Fonctions demandées

L'intégration de PIVAX dans d'autres systèmes demande (1) une adaptation du côté PIVAX, pour l'ajout d'une interface adaptée aux interfaces des autres systèmes, (2) une adaptation éventuelle du côté des systèmes, et (3) la définition d'un format d'échange entre PIVAX et les systèmes concernés.

8.3.1.1. Dictionnaire minimal pour les données dédiées

Dans un système conçu pour la traduction ou la post-édition collaborative comme SECTra_w, l'interface est déjà chargée par les fonctions de support. Si l'on veut ajouter une l'interface de dictionnaire, il faut se limiter à une interface réduite de PIVAX, avec uniquement les informations et les fonctionnalités nécessaires (correspondances dans la langue cible), et les boutons pour voir ou éditer le détail dans l'interface complète de PIVAX, voter ou marquer que ce mot est utilisé ou a une mauvaise qualité dans ce contexte de traduction.

Pour faciliter l'utilisation des ressources lexicales, on a prévu deux parties lexicales dans l'interface de SECTra_w, l'une pour des ressources récupérées par Robodico, l'autre pour les ressources « finalisées », à savoir les traductions des mots et des termes qui sont présents dans des traductions (de segments) de qualité suffisante (pas les sorties brutes de TA, mais les post-éditions). Parmi ces traductions, on peut encore distinguer celles qui sont « recommandées » pour le projet en cours (score au-dessus d'un certain seuil).

Par l'exemple, quand l'utilisateur post-édite un segment, en bas de l'interface de SECTra_w, il voit immédiatement les traductions des mots dans ce segment (Figure 76).

Évolution de PIVAX de la TA hétérogène à la TAO hétérogène

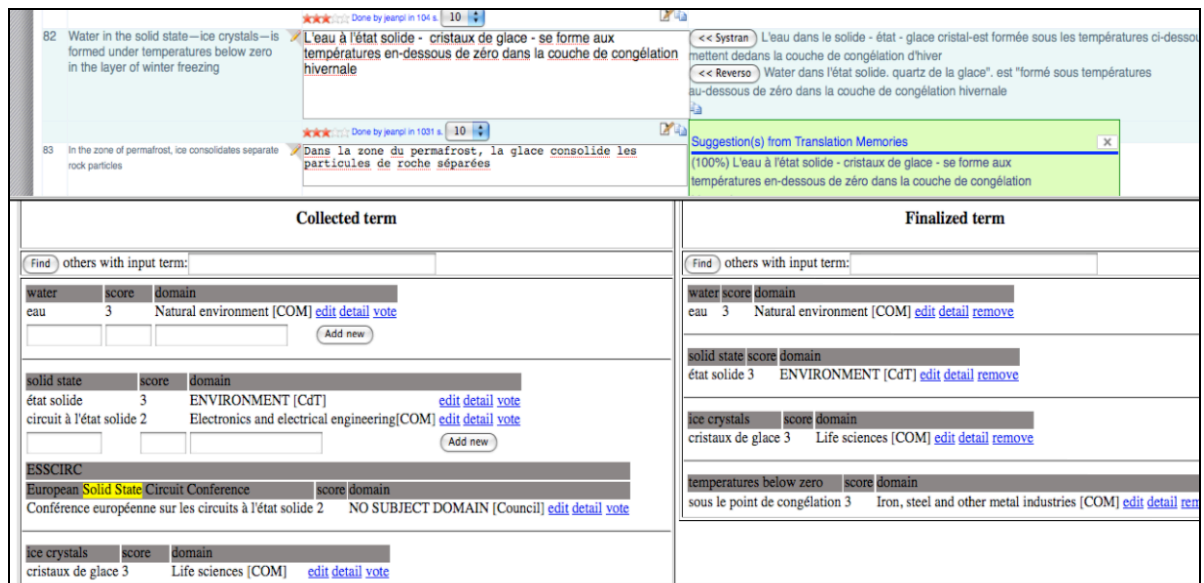


Figure 76: Interface simplifiée de PIVAX dans SECTra_w

8.3.1.2. Calcul proactif et synchronisation

Pour l'instant, avec l'implémentation actuelle de Jibiki, le temps de consultation d'une occurrence est long (10 secondes), et le temps total est environ 2 minutes pour un segment moyen de 12 occurrences.

Pour résoudre ce problème, on utilise le principe de proactivité, qui dit que la consultation (et la TA) doit être faite avant l'accès à un segment. Une fois un document à traduire importé, SECTra_w demande à PIVAX de construire un petit dictionnaire pour chaque segment. Le résultat (un fichier HTML pour chaque segment) est stocké dans SECTra_w. Quand l'utilisateur passe sur un segment, la consultation est déjà prête. Dans le résultat, sur chaque article, on a les correspondances et les liens vers PIVAX pour afficher en détail cet article, et un bouton pour aller éditer sous PIVAX. Une fois qu'un article dans la partie de Robodico a été édité par un humain, il est transféré dans la partie finalisée.

Selon le contexte (couple de langues à traduire, seuil accepté pour afficher les entrées sur l'interface de post-édition...), SECTra_w utilise une CSS pour filtrer et choisir les informations à afficher, et montre immédiatement le résultat de la consultation quand l'utilisateur post-édite ce segment.

Le calcul proactif résout le problème de délai, mais il crée autre instance des données de PIVAX du côté de SECTra_w. Il faut prévoir une synchronisation automatique (périodique) ou manuelle (sur demande d'utilisateur) entre ces données et la base centrale de PIVAX.

8.3.1.3. Échange à la WICALE et évolution vers l'architecture en agents à gros gains

Actuellement, l'échange entre PIVAX et les autres systèmes comme SECTra_w se fait par le protocole http et le format est la page HTML de résultat. Tous les paramètres de la requête http et le format du résultat HTML sont bien conventionnés entre PIVAX et SECTra-w.

Pour communiquer avec plusieurs systèmes à distance, une solution à la WICALE (voir 4.1.2) est pratique dans cette situation. Pour un système, on définit les commandes et le format des résultats associés dans un métalangage simple. Si l'on a un autre système ou s'il y a un changement dans la syntaxe de commande d'échange ou de format, on s'adapte très vite.

Par exemple, on peut proposer un fichier de paramètres sur SECTra_w par la déclaration suivante d'une commande de recherche :

Paramètre	Nom	Valeur par défaut	Note
Protocole	Protocole	http	Protocole de commande
Nom_serveur	server	javaliq.imag.fr	Nom de serveur
Port	port	80	Port
URL	url	/pivax?QuickSearch.po	Chemin sur le serveur
Mot à chercher	headword	'test'	Mot ou suite de mots à consulter
Délai	delai	100	Temps d'attente maximale, s'il le dépasse on passe à une autre requête
Lemma	lemma	yes	Demander de passer à la lemmatisation si elle existe
Segment_ID	segid		Identificateur de segment pour que SECTra_w réattache le résultat à ce segment
Langue_source	ls	en	Langue source
Langue_cible	lc	*	Liste des langues cible à consulter, '*' pour toutes

À partir de cette déclaration, on peut générer la requête suivante avec les valeurs associées pour un segment :

<http://javaliq.imag.fr/pivax/QuickSearch.po?headword=test&delai=100&lemma=yes&segid=10002&ls=en&lc=fr>

Le résultat renvoyé par PIVAX est un fichier HTML sous le format simple de tableau. Sur ce fichier, on déclare les informations à extraire (ex. par XPath) et l'affiche sur l'interface simplifiée de PIVAX dans SECTra_w.

<pre><html> <body> <table> <tr><td lang="en">test</td></tr> <tr><td lang="fr" cat="verb">tester</td></tr> <tr><td lang="fr" cat="noun">épreuve</td></tr> <tr><td lang="fr" cat="noun">essai</td></tr> <tr><td lang="fr" cat="noun">test</td></tr> </table> <div>... autres informations ...</div> </body> </html></pre>		
MOT_SOURCE	/html/body/table/tr/td[lang="en"]	Récupération du mot source
MOT_CIBLE	/html/body/table/tr/td[lang="fr"]	Récupération de la liste des équivalents dans les langues cible

En plus, la partie de communication et de synchronisation entre les données temporaires situées sur ces systèmes et PIVAX doit être indépendante des systèmes connectés et il ne faut surtout pas avoir à demander aux développeurs de ces systèmes de les réaliser. Il s'agit d'un module dédié de type agent implanté sur chaque système, qui s'occupe d'échanger les informations concernées.

8.3.2. Application dans des projets réels de traduction

L'expérience de la THAM montre que la post-édition peut augmenter de 20% à 60% la vitesse de traduction [Allen, 2006] par rapport à la traduction traditionnelle (1h/page standard de 250 mots). Cependant, cette différence n'est pas suffisante pour une contribution lexicale de plus en général (système de THAM ITS-1, voir 2.2.2.1). En augmentant la facilité d'utilisation de PIVAX, on aimerait atteindre une efficacité de post-édition et de contribution lexicale de 30% à 70%. Cela semble être le cas, d'après les évaluations de PIVAX dans les projets de traduction EOLSS/UNL-FR et iMAG.

8.3.2.1. EOLSS/UNL-FR

Rappelons que ce projet avait pour but de traduire 25 articles de l'EOLSS, qui contiennent 200-220K mots, en 9 mois (mars 2008-décembre 2008) par TA suivie de post-édition par des contributeurs bilingues, de langue maternelle française, mais non traducteurs professionnels. Au début du projet, on voulait insérer le développement lexical lors de la traduction. Pour

cela, on a cherché la traduction des mots et termes de ces 25 articles par consultation automatique du site IATE par Robodico. On a obtenu 255K termes en anglais et 258K en français. Puis on propose ces pré-traductions des mots sur chaque segment que l'utilisateur post-éditait dans l'interface de SECTra_w.

Dans SECTra_w, le temps de post-édition pour chaque segment est mesuré par un chronomètre entre le première clic sur ce segment et le clic suivant hors de ce segment. Sur cette mesure, on suit la quantité de la post-édition de chaque utilisateur. Voici les trois meilleurs :

Utilisateur	Mots	Nb. de pages (250 mots)	Minutes /page	Gain (par rapport avec la traduction standard (1h/page))
Contributeur 1 (CB)	80 538	322	5	92 %
Contributeur 1 (NS)	63 948	258	8	87 %
Contributeur 1 (JPG)	24 624	98,5	35	42 %

Ce phénomène trop optimiste et la diversité des résultats peuvent être expliqués par le mécanisme du compteur et la consultation de dictionnaire.

SECTra_w ne compte pas (et ne peut pas compter) le temps total qu'on passe, mais seulement le temps entre le clic initial (on a souvent déjà lu le texte) et le clic de sortie. Le temps entre 2 segments n'est pas compté alors qu'il est souvent non nul. Il peut être du même ordre que le temps mesuré s'il s'agit de segments courts, souvent acceptés tels quels. Il peut être beaucoup plus long si l'on va chercher dans un dictionnaire.

Le temps de recherche dictionnaire n'est pas bien compté, car l'intégration de l'interface simplifiée n'a pas servi comme il faut. En effet, son intégration n'a été effectuée qu'à la fin du projet. Le post-éditeur allait chercher dans des onglets des ses dictionnaires en ligne préférés comme Collins ou Word Reference, voire parfois dans un dictionnaire papier pour certains post-éditeurs (trop) scrupuleux.

Une autre expérience a été faite sur le contributeur 1 avec un chronomètre manuel. Le temps réel de post-édition est environ 15-20 minutes/page après très peu de temps de familiarisation. Quand on connaît le vocabulaire spécifique, on doit effectivement arriver à 5-10 minutes/page. Cependant, cette expérience est très spontanée, il faudrait mesurer en moyenne sur 1h ou 2h d'utilisation réelle et sur un grand nombre de post-éditeurs ayant des profils différents.

8.3.2.2. iMAG

L'accès multilingue à des sites élus via des iMAG (voir 2.1.2.2.2) utilise PIVAX comme base lexicale des mots à traduire pour chaque site en question. Les tâches dans un projet iMAG ressemblent donc à celles réalisées dans SECTra_w.

Sur chaque site à traduire, on segmente les pages HTML en un ensemble de segments, puis on les passe à la lemmatisation, si possible. À partir de la liste de mots obtenus, on lance la consultation automatique par Robodico sur un site de terminologie ou de dictionnaire (IATE, Wikidictionary, ...). Le résultat est importé dans PIVAX.

Une différence avec EOLSS/UNL-FR est qu'on a des sites ayant certaines pages bilingues ou multilingues déjà traduites par un humain. Pour initialiser la MT de l'iMAG, il est nécessaire de récupérer les segments bilingues et aussi les termes bilingues existants dans ces pages. Certains outils d'extraction de termes bilingues (voir 8.2.3.2) peuvent être utilisés. On peut les appeler comme un service extérieur comme Robodico.

Lors de la post-édition d'une page, quand l'utilisateur édite un segment à la volée, on affiche la traduction de ses formes dans un panneau en bas de page montrant deux interfaces simplifiées de PIVAX pour deux dictionnaires.

Cependant, le travail d'un modérateur-animateur pour une langue cible sur un site iMAG mélange entre post-édition, aide aux contributeurs et interaction avec le gestionnaire du site

élu. En plus, les utilisateurs (visiteurs) ne post-éditent que de façon épisodique et opportuniste, avec une quotité relativement limitée (ex. 2 ou 3 heures par semaine). L'évaluation devient plus compliquée, mais avec le support de la consultation dictionnaire de PIVAX, nous évaluons la vitesse d'iMAG à environ 20-25 minutes/page (250 mots).

8.3.3. Extension de PIVAX pour supporter des bases « préterminologiques »

8.3.3.1. Motivation

La motivation d'une base de données lexicales multilingue préterminologique (BDLMpT) vient du fait que le coût de construction d'une base terminologique de bonne qualité et quantité est très élevé, car cela demande beaucoup de travail de personnes disposant de connaissances approfondies dans le domaine concerné [Cabré, Sager et DeCesaris, 1999].

Une solution inspirée de Wikipédia a été proposée, qui est de profiter de la contribution d'utilisateurs bénévoles sur le Web, comme IToldU [Bellynck, 2009], Yakushite [Yakushite.Net, 2008] ou Papillon [Papillon, 2009].

Il faut initialiser la base avec les ressources disponibles quand l'utilisateur commence à développer la base, comme cela a été fait par les projets MultiMatch [Jones et al., 2008], PanImage [Etzioni et al., 2007], Wikitionaries [Wiktionary, 2009], etc.

Une solution, proposée par M. Daoud dans son travail de thèse en cours, consiste à remplacer le but impossible de créer une base terminologique de qualité par une base « préterminologique » ne contenant que ce que des contributeurs non-linguistes et non-terminologues peuvent contribuer, tant en type d'information (au maximum, classe morphosyntaxique, domaine, et exemple d'usage) qu'en exactitude terminologique.

On passe d'abord en revue les systèmes existants de type BDLMpT, puis on approfondit la distinction entre la construction d'une BDLMpT et celle d'une BDLM. Finalement, on montre dans la troisième partie l'application de PIVAX à une expérience d'une telle construction.

8.3.3.2. Projets et travaux en cours

8.3.3.2.1. Le projet DSR et la thèse de M. Daoud

Le projet DSR (Digital Silk Road) au NII-Tokyo depuis 2002 [Ono et al., 2008] a comme sous-projet la multilinguisation d'une grande archive culturelle historique concernant la Route de la Soie, et en particulier de 95 livres rares en 10 langues de la bibliothèque Toyo Bunko.

Dans le cadre de ce projet, M. Daoud cherche à collecter des ressources « préterminologiques » à partir de ressources disponibles sur le Web comme Yahoo Term, Wikipedia Translator..., puis il propose aux lexicographes et terminologues de contribuer sous un environnement de contribution lexicale collaborative sur le Web, CLIR (Cross-Lingual Information Retrieval) [Daoud, Daoud et Boitet, 2009]. Ces ressources récupérées sont conceptuellement un réseau lexical.

L'utilisation de PIVAX est prévue pour créer une telle base. Pour cela, nous avons effectué avec une adaptation minimale : ajout dans la microstructure de l'information de domaine, des exemples d'usage et d'un score de contribution.

8.3.3.2.2. IToldU, une base développée en contexte d'enseignement

Le système IToldU (Interactive Technical On Line Dictionary for Universities) [Bellynck, Boitet et Kenwright, 2005] a été développé pour développer des dictionnaires techniques anglais-français dans le contexte pédagogique de l'EFPG (École Française de Papeterie et Industries Graphiques) de l'INPG. La communauté des utilisateurs contenait environ 250 étudiants par an, répartis en 15 classes. Le domaine technique spécifique recouvre les sciences

liées à l'étude du papier et de la pulpe, les techniques de transformation et d'emballage, la rhéologie, l'impression numérique, et la gestion de la couleur.

Le but d'IToldU est d'aider l'enseignement de l'anglais technique à l'EFPG. Le scénario est que chaque étudiant doit collecter ou créer des termes avec l'information de mot-vedette, définition, contexte et un score dans son dictionnaire personnel depuis n'importe où, grâce à une interface Web avec mot de passe. L'enseignant d'une classe forme des groupes et affecte des mots et expressions aux groupes. Les étudiants dans le groupe s'occupent de traduire et d'introduire dans la base les entrées correspondantes. L'enseignant peut noter les contributions de chaque étudiant, depuis une interface réservée aux professeurs.

L'expérience d'IToldU a été un vrai succès, avec une participation très active des étudiants, qui ont créé une base préterminologique avec une quantité d'environ 17.000 entrées en 2003-2004, 6000 de plus en 2004-2005 (2 fois moins de travail avait été demandé par les professeurs d'anglais, avec en compensation une meilleure qualité des exemples d'usage).

Une des extensions d'IToldU est de le transformer vers un environnement de contribution lexicale multilingue à la Jibiki, en étendant PIVAX pour les flots de contrôle, et en introduisant des interfaces spécifiques (pour les étudiants et pour les professeurs).

8.3.3.3. Réalisation dans PIVAX

Au niveau de la conception, avec la généricité de la macrostructure et la variété des microstructures, PIVAX peut déjà supporter la construction d'une BDLMpT.

Cependant, au niveau pratique, pour une meilleure adaptation aux utilisateurs des BDLMpT dans un contexte spécifique (ex. l'interface pour chaque type d'utilisateur selon leurs tâches comme dans IToldU), il a fallu enrichir PIVAX de fonctions et d'interfaces spécifiques.

8.3.3.3.1. Modification effectuée sur PIVAX

L'extension de la microstructure de PIVAX pour supporter une BDLMpT est relativement simple : on y a simplement ajouté aux champs public mot-vedette, définition, exemples, note et domaine. La macrostructure est identique à celle de PIVAX, avec les liens directs maintenus aux axèmes et axes. Voici un exemple du terme 'motor' :

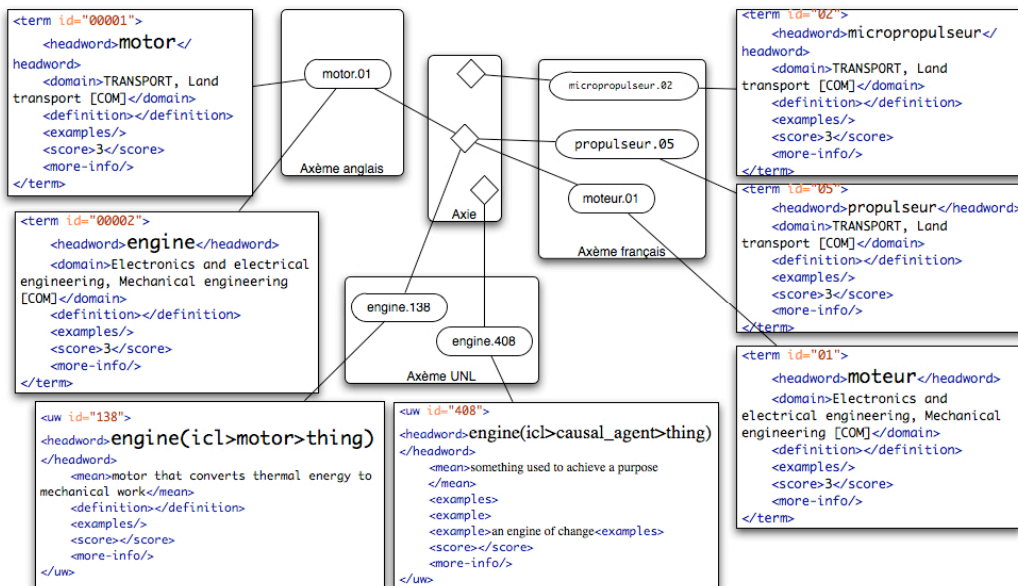


Figure 77: Exemple du mot 'motor' dans une BDLMpT

À partir de cette structure, on peut importer les ressources récupérées (projet DSR, dictionnaire donné par iToldU...) et proposer aux utilisateurs de contribuer. Le temps de définition des structures et l'import des données est environ 1 journée.

8.3.3.3.2. Organisation d'une BDLMpT par rapport aux niveaux de confiance

Une autre différence entre une BDLMpT et une BDLM est que la confiance qu'on peut accorder aux données est très variable. Elles peuvent avoir été obtenues automatiquement par Robodico, ou être en cours de développement par des bénévoles non professionnels dont les contributions ne sont pas certifiées, ou encore elles peuvent être plus ou moins révisées par des experts des domaines associés.

Pour mettre en service une BDLMpT supporté par PIVAX, on peut mettre chaque type de ressource sur un volume séparé, l'origine et le statut des données étant alors connues de façon implicite.

Pour rendre ces informations explicites, on a ajouté aux méta-données du volume de PIVAX un champ de pertinence de ses données. Voici un exemple d'entête d'un volume pour les données « récupérées », « en développement », « révisées » et « certifiées » :

```
<volume-metadata
  location="local" category="multilingual"
  creation-date="28/05/2009 00:00:00" encoding="UTF-8"
  format="xml" installation-date="28/05/2009"
  name="Pivax_iToldU-eng" dictname="Pivax"
  dbname="pivaxclef09eng" version="1"
  source-language="eng" status="development" origine="iToldU"
  <comments>Volume anglais pour iToldU </comments>
  <cdm-elements> ... </cdm-elements>
  ...
</volume-metadata>
```

L'indication de l'origine et du statut des ressources sur chaque volume permet de gérer les flux de données lexicales entre les volumes dans un seul dictionnaire PIVAX.

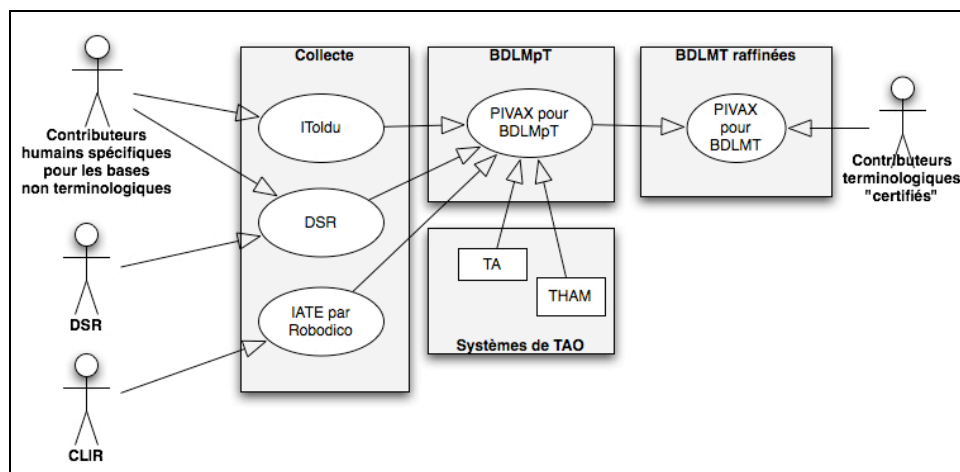


Figure 78: Diagramme de cas d'usage de BDLMpT avec plusieurs volumes de données dans PIVAX

En plus, le mode d'édition à la Wiki sur chaque contribution permet de conserver tous les changements sur les données. Ces deux aspects répondent totalement aux besoins de la conception d'une BDLMpT.

8.3.3.3. *Perspective de la spécification d'interface et support du flot de travaux*

En pratique, la contribution dans une BDLMpT varie selon le projet (ex. DSR, extraction de termes en générique [Ramisch, 2009], produits alimentaires [Ibanescu, Buche et Dibié-Barthélemy, 2009], *etc.*) et selon le contexte (IToldU pour l'enseignement). Chaque base a une interface et des fonctions spécifiques (les contributions des étudiants dans IToldU sont notées par les professeurs, les contributions dans DSR sont révisées par un expert dans le domaine de la culture et de l'histoire de la Route de la Soie, *etc.*).

Dans chaque projet de type BDLMpT, les contributions lexicales suivent des processus différents, mais décomposables en tâches élémentaires en petit nombre. On peut utiliser un outil de gestion de flots de travaux [Tesconi et al., 2006] pour décrire ces projets et spécifier l'interface et les fonctions associées, puis les adapter et les intégrer à PIVAX.

Conclusion

Dans ce chapitre, on a transformé PIVAX en une base lexicale contributive avec les supports nécessaires pour le développement lexical sous son environnement. On a vu également comment intégrer PIVAX dans des systèmes de THAM (SECTra_w, iMAG) pour gérer leur base lexicale, et avons expérimenté cela dans un projet réel de traduction (EOLSS/UNL-FR).

Grâce à ce travail, PIVAX devient un méta-EDL permettant de gérer la partie lexicale des systèmes de TA hétérogènes.

En ce qui concerne les ressources stockées dans PIVAX, on voudrait que PIVAX fonctionne comme un vrai EDL dans lequel des utilisateurs non-informaticiens comme les administrateurs (linguistes ou gestionnaires) de projets de création de BDLMpT pourraient définir eux-mêmes les flots de contrôle, et programmer certaines tâches (à partir d'opérations élémentaires) à un niveau de programmation élémentaire suffisant, à définir. Cet aspect fait l'objet du chapitre suivant, mais son étude n'a pu être qu'entamée dans le cadre de cette thèse.

Chapitre 9. Vers la programmabilité des bases lexicales PIVAX

Introduction

L'idée d'utiliser des Langages Spécialisés pour la Programmation Linguistique (LSPL) a été discutée au 3.2.3. Avec tous les avantages de spécification et d'adéquation, d'autonomie au sens que l'utilisateur peut définir lui-même des traitements selon ses besoins et ses désirs, on désire s'inspirer de cette idée dans le développement lexical. On voudrait qu'une BDLM fonctionne comme un EDL avec certains LSPLex (Langage Spécialisé pour la Programmation Lexicale) pour programmer des tâches spécifiques.

Le désir d'avoir des LSPLex provient également de notre expérience concrète : chaque fois que nous avons eu besoin de produire des données de notre base centrale PIVAX avec un format de sortie spécifique, par exemple la sortie des UW d'U++ sous la forme de règles-Q, ou la génération de l'interface simplifiée pour le dictionnaire d'un segment dans SECTra_w, on a dû programmer en Java ou Velocity un module informatique spécifique. On aimerait faciliter ce travail et libérer les administrateurs linguistiques de l'indépendance d'un informaticien en leur permettant de « programmer » eux-mêmes.

Une autre approche pour résoudre le problème de la variété des opérations sur les données est de proposer un algorithme global pour couvrir le mieux possible les cas de données pour plusieurs langues dans une tâche précise, ex. l'unification des données lexicales. Cette approche ne peut être réalisée qu'à la condition d'avoir PIVAX, une BDLM commune.

Dans ce chapitre, on essaie d'abord de caractériser les niveaux de programmations nécessaires dans une BDLM. Ensuite, on propose des prototypes d'un langage narratif pour la manipulation directe des graphes lexicaux de PIVAX pour illustrer l'approche par LSPLex. Enfin, on propose un premier algorithme global pour unifier les ressources lexicales, appliqué sur le projet U++C.

9.1. Niveaux de programmation souhaités

Le but final du projet PIVAX est de construire un serveur lexical, qui serve à plusieurs systèmes de TAO et à plusieurs types d'utilisateurs avec des besoins différents.

La programmabilité peut être apportée à PIVAX au moins à 3 niveaux : le support au développement contributif des données, la programmation d'algorithmes sur les données, et la programmation de tâches au niveau du système.

9.1.1. Support du développement humain

On souhaite premièrement avoir des paramétrages possibles pour des tâches communes dans la gestion des contributions. De plus, vu variété des fonctions demandées dans chaque application, on ne cherche pas à implémenter toutes les fonctions spécifiques possibles, mais on vise à utiliser l'approche par LSPL pour que les « développeurs-lexicographes » puissent définir eux-mêmes des tâches.

9.1.1.1. Gestion des contributions

Avec l'implémentation actuelle de Jibiki, on peut gérer les groupes et les individus. Cependant, il nous manque des fonctions pratiques pour la traduction comme le suivi des contributions, notions et projets et en particulier la gestion des flots de travaux.

9.1.1.1.1. *Notions de projet, de sous-projet*

La notion de projet et de sous-projet n'existe pas encore dans PIVAX ; nous prévoyons de l'ajouter. Un projet peut relier des données et des personnes sur des tâches, pendant une durée déterminée. Le gestionnaire d'un projet peut suivre le projet en surveillant le processus de développement d'utilisateur.

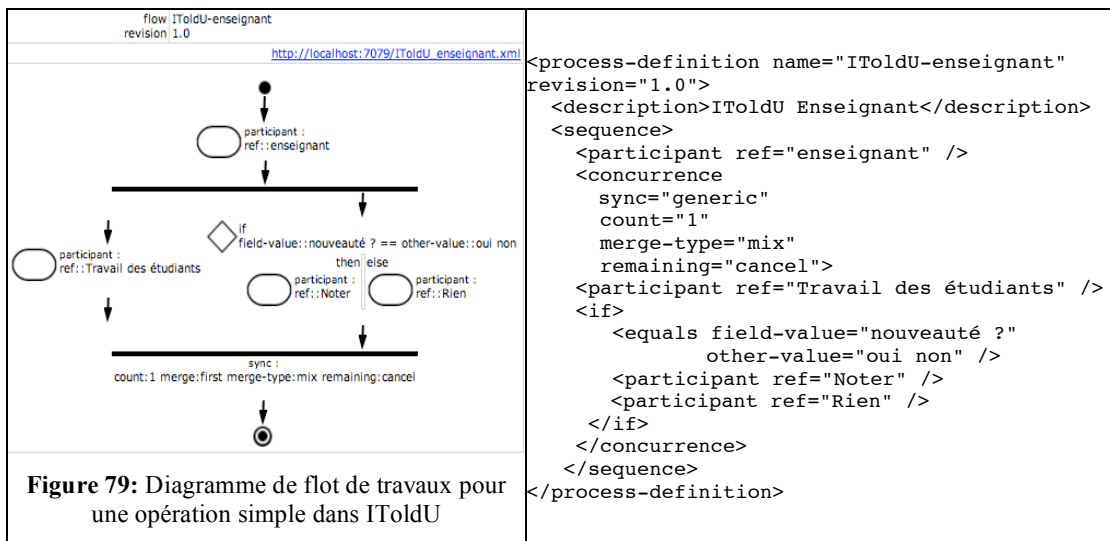
9.1.1.1.2. *Suivi des contributions*

Pour l'instant, on ne peut pas décrire le niveau de compétence professionnel d'un utilisateur. On aimerait qu'un gestionnaire d'un projet puisse mettre un niveau (-1 jusqu'à 5, le 5 est pour un lexicographe certifié) pour un utilisateur sur un projet. À partir de ces niveaux, on pourrait mesurer la qualité et la quantité du travail d'un utilisateur rémunéré et le payer en conséquence.

9.1.1.1.3. *Flot de travaux*

Le développement lexical peut être distribué à plusieurs étapes (extraction du corpus, récupération du Web, correction automatique ou semi-automatique, correction collaborative, validation par les experts, révision dans un corpus, application à la TA...) et ce processus est également une étape dans une application comme développer un nouveau couple de langues de TA. Ce travail peut être répété dans un cycle de développement.

Dans cette situation, on peut utiliser un outil de flot de travaux spécifique pour développer les ressources lexicales. Nous avons expérimenté cette idée avec l'outil Open Workflow [OpenWFE, 2009]. Voici un exemple de la description du flot de travaux pour un enseignant dans IToldU.



9.1.1.2. *Langage narratif de programmation pour des tâches simples*

L'idée d'avoir un langage narratif vient des limites d'interface graphique figée. L'utilisateur ne peut pas réaliser tous ses besoins et est dépendant de l'interface donnée par le développeur. Par exemple, dans la fonction de recherche avancée de Jibiki, qui est déjà assez puissante, on peut définir plusieurs critères de recherche (Figure 80) :

Figure 80: Recherche avancée par plusieurs critères

Cependant, on ne peut pas définir une expression comme ((HEADWORD est ‘abandon’) -ET- (POS est ‘VERB’) -OU- auteur est ‘nguyenht’) via cette interface. Il est clair que le développeur ne peut pas satisfaire tous les besoins spécifiques de tous les utilisateurs.

Construire un langage narratif est une première étape vers un LSPL destiné à développer les ressources lexicales dans PIVAX. L'utilisateur a un moyen plus puissant que les formulaires graphiques (déjà limités par la technologie Web par rapport au WYSIWYG).

On peut proposer les langages narratifs pour supporter des tâches simples en reliant avec la présentation graphique des objets.

Un *langage narratif* [Bellynck, 2001] est un langage où les objets et les actions primitifs sont ceux de l'interface graphique. Il offre les structures de contrôle minimales permettant de réaliser les actions visées. Un programme dans tel langage est une suite d'actions optimisées permettant de produire le résultat souhaité à partir d'un certain état initial.

9.1.2. Programmation possible sur les données lexicales

En parallèle à l'ajout de fonctions, on devrait proposer un moyen aux utilisateurs pour manipuler leurs données. On peut classifier les fonctions selon la complexité de réalisation : (1) l'intégration directe des bibliothèques, (2) la proposition d'une manipulation directe intuitive sur les données, (3) la construction d'un algorithme global pour supporter une tâche précise, et (4) la programmation primitive par un langage.

9.1.2.1. Intégration de bibliothèques adaptées

Un premier niveau est de traiter les données par des appels à des bibliothèques adéquates disponibles, comme MuLLinG (Multi-Level Linguistic Graph) pour la programmation sur des graphes lexicaux ou Jimini pour l'évaluation d'une BDLM.

MuLLinG. La bibliothèque MuLLinG par V. Archer proposée dans sa thèse [Archer, 2009] au GETALP a été définie pour généraliser les programmes *ad hoc* construits pour diverses recherches en extraction de connaissances lexicales. Elle repose sur un modèle de graphes linguistiques à niveaux, étiquetés par des attributs libres, et peut servir à programmer bien autre chose que des algorithmes d'extraction de collocations. Un exemple d'application à la pondération d'équivalents dans un dictionnaire bilingue a d'ailleurs été donné par V. Archer.

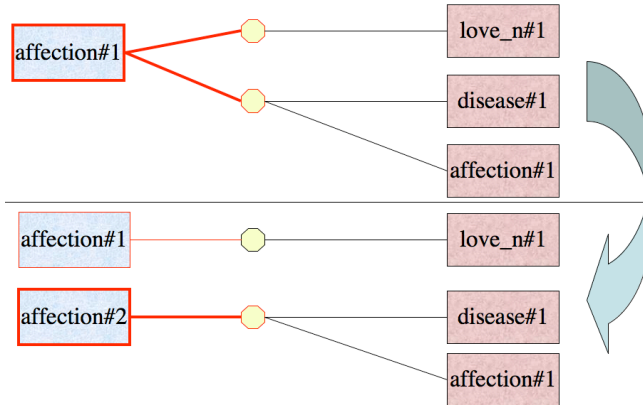
Jemini. Dans le cadre de sa thèse [Teeraparbserree, 2005], A. Teeraparbserree a proposé une plate-forme, Jemini, pour évaluer la qualité des bases de données lexicales. On peut l'utiliser pour valider le travail de développement lexical humain.

9.1.2.2. Primitives de base comme actions de manipulation directe du réseau lexical

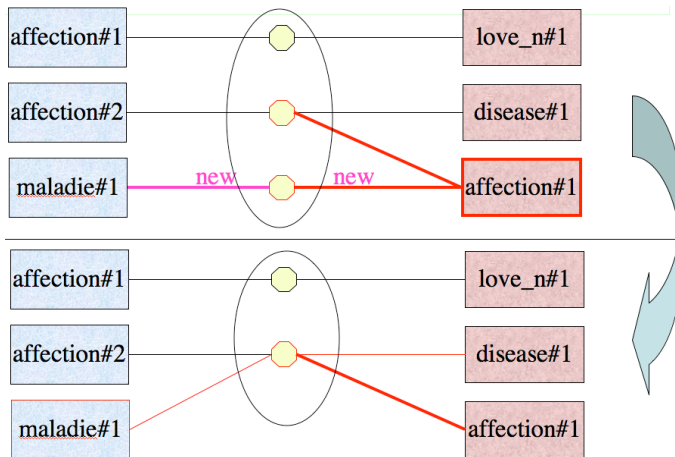
Pour manipuler le graphe, [Boitet, 2005] a proposé des primitives pour restaurer la cohérence dans une BDLM de type Papillon-NADIA. Voici un exemple, dans lequel la contrainte de cohérence est : une lexie ne peut pas être connectée à la fois à 2 axèmes ou axes. Si l'on trouve ce cas de figure, 3 actions sont possibles. Un lexicographe en charge de la base pourra alors définir 3 règles de réécriture sur ce graphe, partageant la même « partie gauche »

(condition, ici représentée en haut), avec des « parties droites » différentes (actions, présentées ci-dessous). Le système devrait alors proposer ces 3 possibilités au contributeur travaillant sur la cohérence de la base et rencontrant une instance de cette partie gauche. « Programmer » consisterait par exemple ici à sélectionner une partie du graphe, puis à lancer un processus de recherche automatique et d'interaction avec l'utilisateur pour décider quelle règle appliquer.

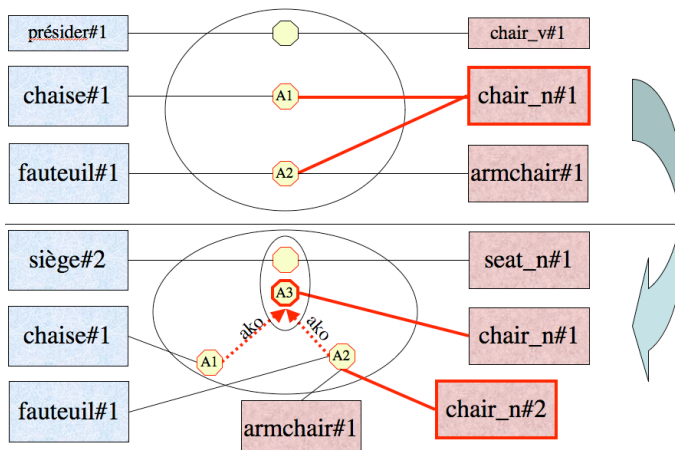
- Fission de lexie ($L1-(A1,A2) \rightarrow L1-A1; L2-A2$):



- Fusion de deux axes ($L1-(A1, A2) \rightarrow L1-(A1+A2)$):



- Création d'une autre axe et d'une lexie de plus ($L1-(A1, A2) \rightarrow L1-A3; L2-A2$; ako((A1, A2), A3)):



Ces primitives pourraient être intégrées directement dans l'interface de PIVAX. On pourrait aussi lui ajouter des actions plus élémentaires, activables sur chaque nœud (lexie, axème, axe) ou arc, comme : créer, supprimer, relier, séparer, fusionner...

9.1.2.3. Langage narratif pour la programmation par des non-informaticiens

Une autre possibilité pour que l'utilisateur puisse manipuler directement ses données sur une BDLM est de lui fournir un langage narratif pour décrire intuitivement sous forme textuelle une séquence complexe d'opérations réalisables par manipulation graphique directe. Le concept a été introduit par [Bellynck, 1999] dans sa thèse sur Cabri-II, en vue de concevoir des activités favorisant l'assimilation de concepts mathématiques.

L'intérêt d'un tel langage est qu'il offre un compromis entre la manipulation directe par l'interface graphique et la programmation. L'interface de manipulation directe est intuitive, mais l'utilisateur est limité par les fonctions données par l'interface. La programmation par un LSPL est puissante, mais abstraite et demande des connaissances spécialisées. Le langage narratif exprime intuitivement des modélisations et des opérations sur la vue géométrique des objets, et l'utilisateur peut définir lui-même des opérations plus complexes.

9.1.3. Programmation de tâches au niveau du système

Parmi les tâches qu'on souhaite rendre programmables, la synchronisation des données, les traitements en « boucle infinie » par des agents, et la commande à distance d'autres BDLM sont les plus importantes.

9.1.3.1. Synchronisation

Dans le contexte d'utilisation de BDLM dans plusieurs systèmes avec plusieurs versions de données modifiées en local, on a fait effort de collecter des ressources dans une base universelle. Cependant, en raison de l'habitude, de la spécification, de la sécurité ou de la protection de la propriété intellectuelle, chaque groupe de développeurs continue toujours à utiliser ses propres outils spécifiques pour travailler sur sa propre instance de ses ressources. La mise en commun des ressources a des avantages forts, mais cela n'est pas suffisant pour persuader tous les auteurs ou utilisateurs des ressources de passer à un environnement commun.

Un compromis est d'accepter l'existence d'instances différentes et d'avoir une possibilité de synchronisation entre les bases.

Le scénario de synchronisation pourrait être le suivant :

- La synchronisation entre la base centrale et des images en local de parties de l'information. Quand la base est utilisée par plusieurs systèmes, il y a une partie de l'information provenant de la base stockée du côté des systèmes (peut-être avec un mécanisme de boîte aux lettres comme au 5.1.3.2) ou gérée par un outil graphique pour travailler hors ligne. La solution envisagée est de calculer la différence entre des versions et de faire la synchronisation selon cette différence.
- La synchronisation entre PIVAX et d'autres bases ou dictionnaires des systèmes connectés. Dans ce cas, il est (quasi) impossible d'accéder directement aux bases à distance (à cause de la protection des données propriétaires). En plus, ces données sont modifiées par les outils utilisés actuellement sur chaque site, donc il est difficile de déterminer quelles parties d'information il faut synchroniser. Un autre problème est la diversité des opérations possibles sur les données. Par exemple, on peut demander de synchroniser seulement les UW d'U++C ajoutées depuis une certaine date avec la catégorie grammaticale verbe ou adjectif. Cela nous amène à certaines actions primitives comme remplacer, ajouter, fusionner, ... pour les articles à synchroniser.

Pour résoudre ce problème, on envisage une solution en deux étapes.

D'abord, on permet aux développeurs de chaque système de décrire les parties d'information à synchroniser dans un métalangage sur les données en local. À partir de cette description, on reconnaît la structure de données permise et on calcule la différence entre les versions des données à synchroniser.

Ensuite, on permet au développeur de gérer lui-même le module de synchronisation, qui est implanté du côté du système comme un agent. Toutes les interactions entre la base centrale de PIVAX et les bases locales sont faites par ces agents.

9.1.3.2. Traitements en « boucle infinie » par des agents externes

Les traitements en « boucle infinie » sont largement appliqués dans le système d'exploitation. En vue d'un système d'exploitation lexical, PIVAX peut s'en inspirer.

PIVAX est toujours en mode d'attente des commandes. Quand une commande arrive, PIVAX la traite selon sa priorité et renvoie le résultat. L'implémentation par la technologie Web (Enhydra/Java) permet de réaliser facilement cette fonction.

9.1.3.3. Métalangage pour l'échange de données et de commandes pour les agents ou les systèmes lexicaux

Un métalangage comme celui de WICALE ou d'EMEU_w est pratique dans le cas où les commandes et les données d'échange sont différentes entre les systèmes.

Pour l'échange des données lexicales entre les BDLM ou entre une BDLM et les systèmes de TA ou de TAO, il y a des points spécifiques à prendre en compte :

- Taille des données : le chargement d'un gros dictionnaire d'un système de TA (ex. ATLAS-II avec 5M entrées) doit être fait en plusieurs morceaux.
- Distribution des données sur plusieurs systèmes : au contraire du cas de WICALE ou d'EMEU_w, où les linguiciels sont centralisés sur un EDL, les données lexicales dans notre architecture peuvent être distribuées sur plusieurs systèmes et chaque système posséder une version. La détermination de la bonne nouvelle version à synchroniser est encore un problème.

9.2. Un premier langage narratif pour la manipulation directe des graphes lexicaux de PIVAX

Suite aux besoins des langages narratifs de manipulation directe des graphes lexicaux, on propose dans cette partie un premier langage pour créer des graphes lexicaux à partir des données de PIVAX, puis visualiser graphiquement ce graphe (par des outils comme GraphViz) ou manipuler directement ce graphe.

Dans PIVAX, les données lexicales sont déjà conceptuellement des graphes lexicaux avec les relations entre lexies, axèmes, axies. Cependant, si l'on veut développer de façon graphique sur un tel graphe, il faut que l'outil créé à partir du sous-graphe sur lequel on veut travailler (en général, une partie des données de PIVAX) sa « structure interne » et la charge en mémoire. Le principe est le même que pour la partie LSPL d'Ariane-Y : les traitements se font sur la SI, beaucoup plus rapidement qu'en accédant la base de données sous-jacente (Postgres dans le cas de Jibiki et donc de PIVAX), et les modifications des données sont à la fois faites sur la SI en temps réel et répercutées dans la base de données avec un délai, par envoi de commandes de mise à jour à PIVAX, à travers une connexion par socket, ou par une requête http, ou par dépôt de fichiers dans une boîte aux lettres, en utilisant en fait le mécanisme de synchronisation de PIVAX décrit plus haut.

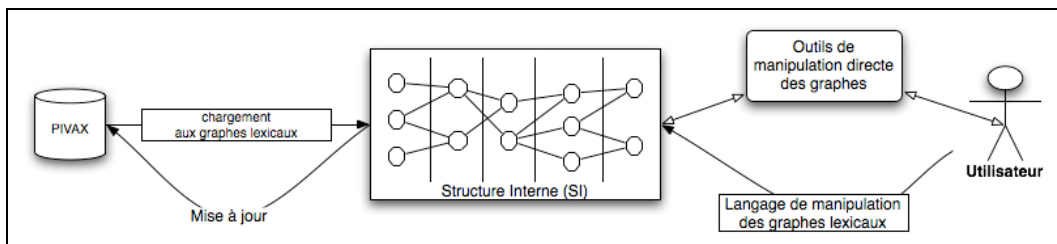


Figure 81: Le processus de manipulation directe des graphes lexicaux dans PIVAX

9.2.1. Spécification

Ce langage est une étape intermédiaire entre la vue graphique et l'organisation interne des objets dans PIVAX. Donc, la syntaxe de ce langage doit être simple, et un « programme » doit se présenter comme suite d'instructions élémentaires sur les objets graphiques manipulés. On s'inspire ici du langage Cabri-programmation introduit par [Bellynck, 1999] dans Cabri-II.

9.2.1.1. Syntaxe

```

Simple :
Créer_nœud (entrée, type, méta-données) ;
Créer_lien (entrée_1, entrée_2, alias) ;
entrée = entrée dans un volume de PIVAX
type = lexie | axème | axie
méta-données = suite_des_attributs_valeurs ;
suite_des_attributs_valeurs (
    ( attribut = valeur ; ) ? attribut = valeur ;
)
Supprimer_nœud (entrée);
Supprimer_lien (entrée_1, entrée_2);

Macro :
NOM_DE_MACRO (parameters) (variables locaux) {
    Suite_des_opérations_simples;
}
Suite_des_opérations_simples {
    (Simple ?;) Simple;
}

```

Dans la syntaxe ci-dessus, il y a deux parties. La première partie concerne les opérations simples pour créer ou supprimer des objets (lexie, axème, axie et les liens entre eux). La deuxième permet de définir des macros pour grouper des suites d'opérations.

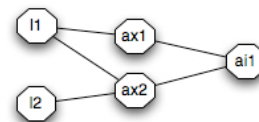
9.2.1.2. Exemples de traitement

Création d'un graphe lexical : on peut créer manuellement un graphe lexical par une suite d'opérations simples. Si on les enregistre, on obtient un programme en PIVAX_lgNarr.

```

Créer_nœud(l1,lexie,(lang='eng'; system='systran'));
Créer_nœud(l1,lexie,(lang='fra'; system='ariane'));
Créer_nœud(ax1, axème, (lang='eng') ;
Créer_nœud(ax2, axème, (lang='fra') ;
Créer_nœud(ai1, axie, (lang='axie') ;
Créer_lien(l1, ax1) ; Créer_lien(l2, ax2) ;
Créer_lien(l1, ax2) ;
Créer_lien(ai1, ax2) ; Créer_lien(ai1, ax1) ;

```



Manipulation directe : On définit ci-dessus trois macros pour traiter les opérations primitives décrites au 9.1.2.2 : la fission d'une lexie en deux lexies, la fusion de deux lexies, la création d'une nouvelle axie et d'une lexie.

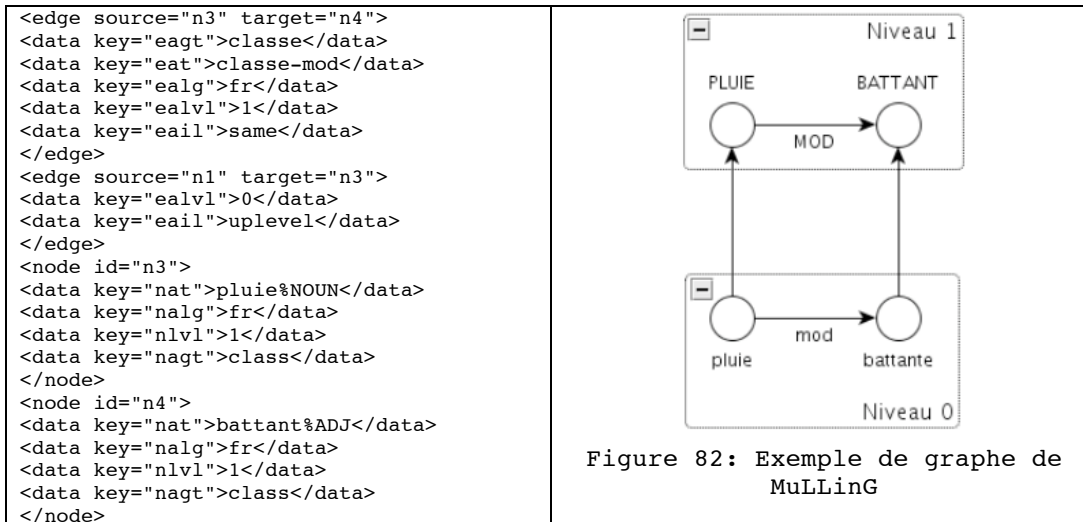
<pre>FISSION(l1, ax1, ax2) (l1'=l1) { Créer_nœud(l1',lexie,(lang='eng'; system='systran')); Supprimer_lien(l1, ax1); Créer_lien(l1',ax1); }</pre>	
<pre>FUSION_AXIE(ax1,ax2,ai1,ai2,ax3,ax4) { Supprimer_lien(ai2,ax2); Supprimer_lien(ai2,ax4); Supprimer_nœud(ai2); Créer_lien(ai1,ax2); Créer_lien(ai1,ax4); }</pre>	
<pre>CREATION_AXIE_AXEME(ax1,ax2,ai1,ai2,ax3,ax4) (ax3' = ax3) { Créer_nœud(ax3',axeme,(lang='eng')); Créer_lien(ai2, ax3'); Supprimer_lien(ai2, ax3); Créer_nœud(ai3,axie); Créer_lien(ai3,ax3); Créer_lien(ai1,ai3,'ako'); Créer_lien(ai2,ai3,'ako'); }</pre>	

9.2.2. Implémentation

9.2.2.1. Représentation par un graphe MuLLinG

Pour représenter le graphe lexical, on voudrait utiliser directement la bibliothèque MuLLinG de V. Archer. Dans cette bibliothèque, la représentation externe des graphes est le langage GraphML [GraphML, 2009]. Par exemple, pour un graphe simple de 2 niveaux :

<pre><graph id="EWN" edgedefault="directed"> <node id="n1"> <data key="nat">pluie</data> <data key="nal">pluie</data> <data key="nap">NOUN</data> <data key="nalg">fr</data> <data key="nlvl">0</data> <data key="nagt">occ</data> </node> <node id="n2"> <data key="nat">battante</data> <data key="nal">battant</data> <data key="nap">ADJ</data> <data key="nalg">fr</data> <data key="nlvl">0</data> <data key="nagt">occ</data> </node> <node id="n3"> <data key="nat">pluie%NOUN</data> <data key="nalg">fr</data> <data key="nlvl">1</data> <data key="nagt">class</data> </node> <node id="n4"> <data key="nat">battant%ADJ</data> <data key="nalg">fr</data> <data key="nlvl">1</data> <data key="nagt">class</data> </node> <edge source="n1" target="n2"> <data key="eagt">mono</data> <data key="eat">mod</data> <data key="ealg">fr</data> <data key="ealvl">0</data> <data key="eail">same</data> </edge> <edge source="n2" target="n4"> <data key="ealvl">0</data> <data key="eail">uplevel</data> </edge> <edge source="n3" target="n4"> <data key="eagt">classe</data> <data key="eat">classe-mod</data> <data key="ealg">fr</data> <data key="ealvl">1</data> <data key="eail">same</data> </edge> </graph></pre>	<pre><edge source="n1" target="n2"> <data key="eagt">mono</data> <data key="eat">mod</data> <data key="ealg">fr</data> <data key="ealvl">0</data> <data key="eail">same</data> </edge> <edge source="n1" target="n3"> <data key="ealvl">0</data> <data key="eail">uplevel</data> </edge> <edge source="n2" target="n4"> <data key="ealvl">0</data> <data key="eail">uplevel</data> </edge> <edge source="n3" target="n4"> <data key="eagt">classe</data> <data key="eat">classe-mod</data> <data key="ealg">fr</data> <data key="ealvl">1</data> <data key="eail">same</data> </edge> </graph></pre>
---	--



Pour manipuler ce graphe, MuLLinG permet de charger ce format à sa structure interne et il propose quelques opérations primitives (ajout/suppression d'un nœud ou d'un arc) et des opérations sophistiquées comme émergence, filtrage, application... sur un ensemble de nœuds. Après le traitement, on peut exporter les données au format externe graphML.

Pour implémenter notre langage PIVAX_IGNarr. On peut suivre deux voies: (1) on peut interpréter de notre langage PIVAX_IGNarr aux primitives de MuLLinG et les exécuter sur un graphe avec le format graphML. Ou (2) on compile le langage PIVAX_IGNarr et on travaille directement sur la structure interne de graphe de MuLLinG, qui est représenté comme une liste d'adjacence des nœuds dans la bibliothèque de gestion de graphe Boost [Boost, 2009] en C++. Dans les deux cas, il nous faut un algorithme pour charger le graphe ou l'exporter au format graphML à partir de notre base PIVAX.

9.2.2.2. Algorithme pour charger le graphe de la base de données

Le chargement des données de PIVAX dans la structure interne définie est réalisé par une recherche exhaustive dans tous les volumes et les relations possibles entre lexie et axème, axème et axie, et entre les axes et les axèmes.

On part du volume pivot axie. Pour chaque entrée dans ce volume, on cherche les entrées reliées dans les volumes d'axèmes dans les langues. Pour chaque entrée trouvée dans chaque volume d'axème, on cherche les lexies connectées à cet axème. Comme il y a des entrées connectées à aucune entrée dans le volume d'axes, il faut marquer les entrées ajoutées dans les volumes d'axèmes et de lexies. Puis, on recherche dans les volumes des axèmes et des lexies pour charger les restes. Voici le pseudo-code de cet algorithme.

```

entrée: base de PIVAX
sortie: l'ensemble des graphes lexicaux

Pour chaque entrée axie dans volume d'axie
  Si pas_encore_visité(axie)
    Créer_graphe(g, axie);
    marquer_visité(axie); créer_noeud(axie);
    liste_des_axemes = prendre_les_axemes_d'une_axie(axie);

    // Traiter les relations entre les axes
    Pour chaque axie_connectée dans liste_des_axies_connectées(axie)
      Si pas_encore_visité(axie_connectée)
        créer_lien(axie, axie_connectée);
    fPour

    Pour chaque axème dans la liste_des_axèmes
      marquer_visité(axème);

```



```

        créer_noeud(axème); créer_lien(axie, axème);
        liste_des_lexies = prendre_les_lexies_d'un_axème(axème);
        Pour chaque lexie dans la liste_des_lexies
            marquer_visité(lexie);
            créer_noeud(lexie); créer_lien(axème, lexie);
        fPour;
    fPour;
fPour;
Charger_des_axème_non_visités();

```

9.2.2.3. Mise à jour différée de la base

Les données sont modifiées dans la mémoire, mais il faut mettre à jour les modifications dans la base de PIVAX. Sur chaque nœud (lexie, axème, axie), on n'envoie que les nœuds modifiés avec l'identification « mulling ». Le format envoyé à PIVAX est similaire au format XML de cette entrée. Voici un exemple d'un axème à envoyer.

Axème initial	Axème après avoir manipulé
<pre> <p:axeme p:id="axeme.unl.access.1"> <p:entryref p:idref="EOLSS.access.1" p:system="EOLSS"> <p:entryref p:idref="UNLDeco.access.14" p:system="UNLDeco"> <p:entryref p:idref="UNLKB.access.2" p:system="UNLKB"> </p:axeme> </pre>	<pre> <p:axeme p:id="axeme.unl.access.1"> <p:entryref p:idref="UNLDeco.access.14" p:system="UNLDeco"> <p:entryref p:idref="EOLSS.access.2" p:system="EOLSS"> </p:axeme> </pre>

PIVAX cherche cet axème dans la base grâce à son identificateur (axeme.unl.access.1), puis compare la différence. Dans cet exemple, PIVAX supprime le lien vers l'article « EOLSS.access.1 » et ajoute un lien vers « EOLSS » de l'article « EOLSS.access.2 » dans le volume EOLSS. PIVAX reindexe incrémentalement chaque article envoyé.

Comme MuLLinG a été implémenté en C++, on utilise un outil compatible de gestion de connexion http comme Neon [Neon, 2009] pour renvoyer la commande de mise à jour à PIVAX. Le morceau de code en C/C++ suivant montre l'ouverture d'une liaison avec PIVAX et l'envoi d'une commande de mise à jour (comme en CASH, ou WICALE). Il y a un autre processus pour récupérer la réponse de PIVAX et l'enregistrer dans un log.

```

/* Création d'une session à PIVAX */
ne_session *sess;
sess = ne_session_create("http", "javalig.imag.fr", 80);

/* Création d'une requête à PIVAX */
request *req = ne_request_create(sess, "MKCOL", "pivax/");

/* Ajout des paramètres à envoyer */
ne_add_request_header (req, "user", "mulling"); //Nom d'utilisateur inscrit dans PIVAX
ne_add_request_header (req, "password", "****"); //Mot de passe
ne_add_request_header (req, "action", "update");//Commande de mise à jour
int id = getNextID(listeID); //Identificateur de la commande
ne_add_request_header (req, "id", id);
ne_add_request_header (req, "entryid", " axeme.unl.access.1");
char *data = "<p:axeme p:id=\"axeme.unl.access.1\">
  <p:entryref p:idref=\"UNLDeco.access.14\"
    p:system=\"UNLDeco\">
  <p:entryref p:idref=\"EOLSS.access.2\"
    p:system=\"EOLSS\">
</p:axeme>";
ne_set_request_body_buffer (req, "content", data);

/* Envoyer à PIVAX et mettre l'identificateur de la requête dans une liste */
if (ne_request_dispatch(req)) {
    printf("Request failed: %s\n", ne_get_error(sess));
} else {
    putID(listeID, id);
}

// Dans un autre processus, on vérifie l'état de cette commande
// et on l'écrit dans un log.
while (getStatut(id) == TRUE) {
    writeLogSucess(id);}

```

9.2.3. Interface de prototype

L'interface de manipulation directe sur le Web (spécifiée mais pas encore implémentée) est illustrée par la Figure 83. On saisit le mot relié dans le graphe. Le résultat de la recherche s'affiche graphiquement dans un panneau à gauche. On peut exécuter des actions élémentaires en cliquant sur des icônes d'objet et d'action. On peut aussi développer un programme en langage narratif dans un champ de texte en bas, puis l'exécuter. Le résultat est affiché à droite. On peut cliquer sur « Mise à jour » pour effectuer cette manipulation dans la base PIVAX.

Figure 83: Interface de manipulation directe des graphes lexicaux par programmation en langage narratif (PIVAX_lgNarr)

9.3. Premier algorithme global pour unifier les UW d'UNL

Pour illustrer l'approche par langage du traitement des données variées et des demandes des fonctions spécifiques dans la programmation, nous proposons ici un algorithme (préliminaire) exécutable sur la SI MuLLinG correspondant à la base de données PIVAX des UW de diverses provenances, et réalisant une unification de ces données par transitivité.

On dispose au départ des dictionnaires d'UW développés séparément par les centres de langue et des UW++ créées par l'UPM et l'IPPI à partir du WordNet anglais (PWN).

9.3.1. Spécification

On a plusieurs volumes monolingues d'UW avec des microstructures différentes. Cependant, chaque UW possède au moins deux informations communes : le mot-vedette (headword) et les restrictions (voir l'exemple au 7.3.3.1.1). Le mot-vedette peut être utilisé pour relier les UW dans les dictionnaires, puis les restrictions peuvent être utilisées pour raffiner le résultat. Pour deux lexies d'UNL ayant le même mot-vedette et les mêmes restrictions, on crée une entrée dans le volume d'axèmes UNL pour relier ces lexies, et une axie pour relier cet axème aux axèmes synonymes des autres langues.

Il faut en pratique aller un peu plus loin qu'une recherche d'identité car, et c'est justement la source de la variété des UW créées par différents groupes pour la même « acception interlingue », la valeur de certaines restrictions peut être plus ou moins spécifique en fonction de chaque lexicographe (`car(icl>animal)` ou `cat(icl>mammal)`) et certaines restrictions peuvent être présentes ou absentes (`(drive(icl>do,agt>human)` ou `drive(icl>do,agt>human,obj>vehicle)`) [Boitet, 2002]. Reprenons l'exemple des UW ayant le même mot-vedette 'access' dans les deux dictionnaires U++C et UNLKB:

U++C	UNLKB
<p>122 : access(icl>recover>do, agt>thing, obj>thing); Cat : VERB; Déf. : obtain or retrieve from a storage device; as of information on a computer;</p> <p>123 : access(icl>reach>do, agt>thing, obj>thing); Cat : VERB; Déf. : reach or gain access to;</p>	<p>177 : right(icl>abstract thing) 178 : access(icl>right) 1972 : way(icl>abstract thing) 1975 : access(icl>way) 5321 : reach(agt>thing,obj>thing) 5323 : access(icl>reach(agt>thing,obj>thing) 6987 : open(agt>thing,obj>thing) 6992 : access(icl>open(agt>thing,obj>thing))</p>

Sémantiquement, les UW numéro '123' et '5323' représentent le même concept, mais les restrictions sont différentes. Pour résoudre ce problème, il faut analyser les restrictions et utiliser d'autres informations (par exemple la hiérarchie donnée dans UNLKB, et les 200K vecteurs conceptuels créés par D. Schwab pour PWN [Schwab et Lim, 2008] afin de calculer des (pseudo)distances entre les UW ayant le même mot-vedette). Dans cet algorithme préliminaire, nous nous limitons à la comparaison exacte des restrictions.

9.3.2. Réalisation

Voici l'algorithme simple d'unification en pseudo code.

```

Entrée: base PIVAX et liste des volumes d'UW à unifier
Sortie: base PIVAX modifiée

/* Unifier les ressources d'UNL sur les volumes d'UNL indiqués */
Unifier(liste_des_volumes_UNL) {
  Pour chaque volume vol de lexies
    Pour chaque entrée uw dans vol
      Pour tous les uw1 dans les autres volumes UNL
        Si Comparer(uw, uw1, niveau) = correspondance
          S'il n'y a pas encore d'axème (uw, uw1)
            axème = Créer_axème(uw, uw1);
          Si le niveau est 2
            Relier_axie(uw, uw1, axème);
        fSi
      fPour
    fPour
  fPour
}

/* Comparer deux UW selon le niveau:
  1: Comparer seulement le mot-vedette
  2: Comparer le mot-vedette et les restrictions
  3: Comparer les autres informations
Paramètres:
  uw: le premier UW
  uw1: le deuxième UW
  niveau: niveau de comparaison
*/
Comparer(uw, uw1, niveau) {
  Niveau = 1 :
    Si uw.mot-vedette = uw1.mot-vedette rendre Correspondant
  Niveau = 2 :

```

```

        Si (uw.mot-vedette=uw1.mot-vedette)
            ET (uw.restrictions = uw1.restrictions)
            rendre Correspondant
    Niveau >= 3 :
        Fonction_Spécifique_pour_comparer(uw,uw1) ;
}

/*
    Relier les deux parties des lexies de LN pour deux UW unifiés
    par une nouvelle axie.
    Paramètres:
    uw: le premier UW
    uw1: le deuxième UW
    axème : axème vient d'être crée pour unifier deux UW
*/
Relier_axie(uw, uw1, axème) {
    Dupliquer les axèmes, les lexies connectés à uw dans les LN;
    Dupliquer les axèmes, les lexies connectés à uw1 dans les LN;
    Créer une nouvelle axie pour relier axème et nouveaux axèmes
    dupliqués
}
    
```

Nous avons implémenté cet algorithme en C++ pour les niveaux 1 et 2 de comparaison des UW.

9.3.3. Résultats

On a expérimenté avec les dictionnaires d'UNL importés dans PIVAX. Voici le résultat.

Niveau		UW communs entre les dictionnaires
Mot-vedette	1	3 245
Mot-vedette et restriction	2	1 425
Spécifique	3	pas encore réalisé

Au départ, la proportion d'UW identiques dans les 6 volumes disponibles est très faible. Si

$$Prog_UW_communes_i = \frac{\#UW_synonyme_i}{\#UW_textuellement_différentes}$$

alors $\#UW_textuellement_différentes = 148\ 639$, on a le tableau suivant :

Ressource	Type/Langue	Nombre des entrées	Taux pour niveau 1
serveur UNLDeco du GETALP	UNL-FRA	39 389	8,23%
PARAX	UNL-FRA	18 978	17,09%
PARAX	UNL-CHN (données par le Pr. SHI Xiao Dong en 2001)	9 315	34,84%
PARAX	UNL-RUS	13 817	23,49%
PARAX	UNL-ESP	3 833	84,66%
UNLKB	UNL	21 618	15,01%
UPM U++C	UW avec définition et exemples	207 009	1,57%
UNDL	Projet EOLSS-UNL-FR/UW	21 354	15,2%

Le nombre des UW communs est faible car il y a un volume (UNL-ESP), qui n'a que 3 833 entrées. Si l'on compare chaque volume d'UW avec le volume le plus gros (U++C, 200K entrées), on a le résultat suivant :

Ressource	Type/Langue	Nombre des entrées	Taux pour niveau 1	Taux pour niveau 2
UNL-Deco	UNL-FRA	39 389	96,67%	47,83%
PARAX	UNL-FRA	18 978	93,98%	44,24%
PARAX	UNL-CHN (données par le Pr. SHI Xiao Dong en 2001)	9 315	99,99%	72,14%
PARAX	UNL-RUS	13 817	91,9%	53,95%
PARAX	UNL-ESP	3 833	99,8%	37,17%
UNLKB	UNL	21 618	93,16%	34,78%
UNDL	Projet EOLSS-UNL-FR/UW	21 354	98,96%	30,86%

Le dictionnaire U++C (200K d'entrées) couvre la plupart des autres dictionnaires. On peut l'utiliser comme un pivot pour créer des dictionnaires LN-UW++. Par l'exemple, on crée le dictionnaire FRA-U++C pour UNLDeco. Tous les UW communs (niveau 2) entre U++C et le dictionnaire UNL-FRA d'UNLDeco ont déjà des codes pour le système UNL-FR écrit en Ariane ($47,83\% \times 39.389 = 18.839$ entrées). Il faut les ajouter pour les UW qui n'ont pas encore de définition ($207\ 009 - 18389 = 188620$ entrées, soit 91,11% à développer).

Conclusion

Dans ce chapitre, on a vu comment ajouter à PIVAX de la programmabilité. Le niveau de la programmabilité n'est pas limité aux API appelées par les autres programmes, mais on introduit la possibilité qu'un utilisateur naïf puisse programmer dans un langage narratif. De plus, grâce à PIVAX, on peut définir des algorithmes pour enrichir ou créer automatiquement des ressources. Une expérimentation a été faite pour unifier les dictionnaires UNL du projet U++C.

Nous arrivons ainsi à faire de PIVAX un EDL pour le développement et l'exploitation de ressources lexicales, par des systèmes de TA hétérogènes aussi bien que par des utilisateurs humains, post-éditeurs contribuant de l'information préterminologique à la volée, ou lexicographes travaillant spécifiquement sur les ressources lexicales.

Synthèse & perspectives

Dans cette thèse, nous avons étudié les problèmes posés par la conception et la réalisation de la partie logicielle des systèmes de TAO hétérogènes, qui se développent à côté des systèmes de TA homogènes et les supplanteront peut-être à moyen terme. Ces nouveaux systèmes intégreront une partie purement TA (traduction automatique), et une partie THAM (traduction humaine aidée par la machine) reposant sur des mémoires de traductions. D'autre part, leurs différents composants seront construits par des équipes différentes, distribuées autour de la planète, avec des méthodes (algorithmiques) et des outils (langages spécialisés ou LSPL, types de dictionnaires, de grammaires, de transducteurs à règles) différents, et des environnements de développement (EDL) différents. Parmi les points intéressants sur lesquels nous avons avancé, on peut noter les quatre suivants.

- *L'amélioration des « méta-EDL de TAO ».* Il s'agit d'une transition incrémentale entre les EDL « natifs » des différents partenaires construisant un système de TA hétérogène, et un futur EDL « intégrateur », ou « universel » dans lequel on « rapatrierait » toutes les fonctionnalités, et en particulier la compilation et l'exécution des LSPL. Partant de CASH, un méta-EDL spécialisé pour l'EDL Ariane-G5 du GETA, et de WICALE 1.0, un premier méta-EDL générique, nous avons montré comment intégrer des possibilités de navigation dans les fichiers source (grammaires, dictionnaires) en réalisant une précompilation en ANTLR permettant de construire des fichiers html « à la Doxygen ».
- *La conception et la réalisation d'une base lexicale pour systèmes de TAO hétérogènes partageant un même pivot lexical.* PIVAX permet de réunir les aspects DA (dictionnaires, pour la TA) et DH (dictionnaires d'usage, pour la TH), et d'avancer dans la résolution du difficile problème de la synchronisation entre un service Web collaboratif comme PIVAX et les environnements locaux de développement lexical comme ceux créés par notre équipe (PARAX et CASH) ou ceux de nos partenaires japonais (UWGate), espagnols (UNL-ES, UPM), ou russes (ETAP-3, IPP1).
- *La réingénierie d'un langage spécialisé « externe » (non supporté par l'EDL Ariane-G5).* Elle a permis de mieux comprendre les problèmes posés par l'implémentation d'un « langage spécialisé pour la programmation linguistique », et les solutions en cours de validation dans le projet Ariane-Y (J.C. Durand, Ch. Boitet). Il s'agit en l'occurrence des systèmes-Q, le fameux LSPL créé par A. Colmerauer en 1967, et qui servit de base pendant 15 ans au système de TA TAUM-météo destiné aux bulletins météorologiques canadiens (30 M mots/an vers 1990, avec moins de 3% de corrections manuelles).
- *La conception et la réalisation d'un « moniteur » adapté à la partie « production » d'un système de TAO hétérogène.* EMEU_w.1.0 a été développé, et est utilisé par l'UNDL dans le cadre de la suite (en cours) du projet EOLSS/UnescoL, qui vise à traduire l'encyclopédie en ligne EOLSS (Encyclopedia of Life Support Systems) depuis l'anglais vers les autres langues de l'Unesco (arabe, chinois, espagnol, français, russe).

À partir des ces composants, nous nous proposons d'intégrer à court terme au système Ariane-Y, en vue d'obtenir un système de TAO hétérogène complet :

- *un moniteur semi-graphique dérivé de WICALE et d'EMEU_w.* Il suffira pour cela de définir concrètement les méta-langages d'échange de commandes et de données dans la nouvelle architecture utilisant REXX et les REXXstacks. Une fois qu'on aura ce méta-EDL, on pourra ajouter des traitements externes (ex. une phase par approche statistique).
- *les systèmes-Q, natifs et étendus.*
- *PIVAX et SECTra_w, dans leurs versions étendues et programmables.*

À long terme, nous continuerons à poursuivre les recherches sur ces différents thèmes en visant l'objectif fondamental : construire des (*méta*) systèmes de TA ou TAO hétérogènes, universels ou génériques. On peut envisager quatre axes.

- *LSPL*. Pour que les utilisateurs non-spécialisés (développeurs linguistes) puissent les utiliser, nous envisageons deux directions. Premièrement, on peut définir et développer des versions simplifiées et plus spécifiques de certaines tâches, comme TTEDIT de J-C. Durand ou l'outil de syllabification SYLLA de C. Del Vigna et V. Berment. L'autre possibilité est de réorganiser un LSPL complexe comme ROBRA de façon à le rendre « appropriable » de façon pédagogique, en activant étape par étape ses différentes fonctionnalités, le premier niveau correspondant par exemple (pour ROBRA) à tous les paramètres par défaut et à une seule grammaire transformationnelle sans itération ni appel récursif ni contexte ni désordre.
- *EDL*. Nous voudrions transformer WICALE en un EDL générique complet, sous forme d'un service Web. Nous permettrions le développement mutualisé des composants linguiciels par la communauté des utilisateurs non-spécialistes et par des spécialistes, un peu comme Wikipedia.
- *BDLM*. La représentation des connaissances lexicales d'un ensemble quelconque de systèmes de TA reste un problème très difficile. Il nous faut étudier et expérimenter d'abord PIVAX pour supporter un certain nombre de dictionnaires de systèmes à transfert, en commençant par ceux disponibles sous Ariane-G5.
- À la côté de la THAM, en parallèle avec les efforts pour faciliter la contribution par des bénévoles et des spécialistes, on cherchera à évaluer en qualité et en quantité le travail lexical, ce qui semble possible sur l'interface directe de PIVAX ou dans un contexte de traduction, sous le contrôle d'un flot de travaux.

Bibliographie

- [Aït-Mokhtar, Chanod et Roux, 2002] **Salah Aït-Mokhtar, Jean-Pierre Chanod et Claude Roux (2002)** *Robustness beyond shallowness: incremental deep parsing*. Natural Language Engineering, Volume 8, Issue 3 (June 2002), Cambridge University Press New York, NY, USA, pp. 121-144.
- [Al-Adhaileh et Tang, 1999] **Mosleh H. Al-Adhaileh et Enya Kong Tang (1999)** *Example-Based Machine Translation Based on the Synchronous SSTC Annotation Schema*. MT Summit VII, September 13-17, 1999, Singapore, pp. 244-249.
- [Al-Adhaileh, 2003] **Mosleh Hmoud Al-Adhaileh (2003)** *Synchronous Structured String-Tree Correspondence (S-SSTC) and its applications for machine translation*. PhD thesis, UTMK, Penang, Universiti Sains Malaysia, 163 p.
- [Albatal, 2005] **Rami Albatal (2005)** *La prise en compte des flux de travaux pour la construction collaborative des bases lexicales multilingues*. Mémoire de M2R, UJF (Grenoble 1), 62 p.
- [ALPAC, 1966] **ALPAC (1966)** *ALPAC Report, "A report by the Automatic Language Processing Advisory Committee"*. Languages & machines. Computers in translation and linguistics. Nat. Academy of Sciences, Nat. Research Council, Washington D.C., 138 p.
- [Alshawi, Buchsbaum et Xia, 1997] **Hiyan Alshawi, Adam L. Buchsbaum et Fei Xia (1997)** *A Comparison of Head Transducers and Transfer for a Limited Domain Translation Application*. European Chapter Meeting of the ACL, Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics, Madrid, Spain, pp. 360-365.
- [Archer, 2009] **Vincent Archer (2009)** *Graphes linguistiques multiniveau pour l'extraction de connaissances : l'exemple des collocations*. Thèse UJF (Grenoble 1), 24 septembre 2009, 274 p.
- [Arnold et al., 1994] **Douglas Arnold, Lorna Balkan, R. Lee Humphreys, Siety Meijer et Louisa Sadler (1994)** *Machine translation: an introductory guide*. Oxford, NCC Blackwell Pub. (January 1994), ISBN-13: 978-1855542174. 200 p.
- [Aymerich, 2004] **Pan American Health Organization: Julia Aymerich (2004)** *Machine Translation in Practice at PAHO, Tutorial Notes*. Sixth Biennial Conference of the Association for Machine Translation in the Americas (AMTA), September 24, 2004, Georgetown University, Washington, DC, 73 p.
- [Bachut, 1994] **Daniel Bachut (1994)** *Le projet EUROLANG : une nouvelle perspective pour les outils d'aide à la traduction*. Proc. TALN-94, journées du PRC-CHM, Marseille, 7-8 avril 1994, Univ. de Marseille.
- [Bachut et Verastegui, 1984] **Daniel Bachut et Nelson Verastegui (1984)** *Software Tools for the Environment of a Computer Aided Translation System*. COLING-1984, July 2-6, 1984, Stanford University, California, USA, pp. 330-333.
- [Bellynck, 1999] **Valérie Bellynck (1999)** *Introduction d'une vue textuelle synchronisée avec la vue géométrique primaire dans Cabri-II*. Thèse UJF (Grenoble 1), 250 p.
- [Bellynck, 2001] **Valérie Bellynck (2001)** *Un langage « narratif » pour Cabri-géomètre*. Hypermédias et Apprentissages 5, Grenoble, 9, 10 et 11 avril 2001, pp. 344-346.
- [Bellynck, Boitet et Kenwright, 2005] **Valérie Bellynck, Christian Boitet et John Kenwright (2005)** *ITOLDU, a Web Service to Pool Technical Lexical Terms in a Learning Environment and Contribute to Multilingual Lexical Databases*. Alexander F. Gelbukh (Ed.):

Computational Linguistics and Intelligent Text Processing, 6th International Conference, CICLing 2005, Mexico City, Mexico, February 13-19, 2005, Proceedings, pp. 324-332.

[Bennett et Slocum, 1985] **Winfield S. Bennett et Jonathan Slocum (1985)** *The LRC machine translation system*. Computational Linguistics, Volume 11, Issue 2-3 (April-September 1985), Special issue on machine translation, pp. 111-121.

[Berment, 2004] **Vincent Berment (2004)** *Méthodes pour informatiser les langues et les groupes de langues « peu dotées »*. Thèse UJF (Grenoble 1), 277 p.

[Bernhard, 2006] **Delphine Bernhard (2006)** *Multilingual Term Extraction from Domain-specific Corpora Using Morphological Structure*. 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL'06), April 3-7, 2006, Trento, Italy, pp. 171-174.

[Besacier, 2007] **Laurent Besacier (2007)** *Transcription enrichie de documents dans un monde multilingue et multimodal*. Mémoire de HDR, UJF (Grenoble 1), 64 p.

[Besacier, Ben-Youssef et Blanchon, 2008] **Laurent Besacier, Atef Ben-Youssef et Hervé Blanchon (2008)** *The LIG Arabic / English Speech Translation System at IWSLT08*. International Workshop on Spoken Language Translation (IWSLT 2008), Waikiki, Hawai'i, USA, 20-21 October 2008, pp. 58-62.

[Bey, 2008] **Youcef Bey (2008)** *Aides informatisées à la traduction collaborative bénévole sur le Web*. thèse UJF (Grenoble 1), 265 p.

[Bey, Kageura et Boitet, 2006] **Youcef Bey, Kyo Kageura et Christian Boitet (2006)** *Data Management in QRLex, an Online Aid System for Volunteer Translators*. Computational Linguistics and Chinese Language Processing, Vol. 11, No. 4, December 2006, pp. 349-376.

[Bey, Kageura et Boitet, 2007] **Youcef Bey, Kyo Kageura et Christian Boitet (2007)** *BEYTrans: A Free Online Collaborative Wiki-Based CAT Environment Designed for Online Translation Communities*. PACLIC21, The 21st meeting Pacific Asia Conference on Language, Information and Computation, November 1-3, 2007, Seoul National University, Korea, 8 p.

[Bey et al., 2008] **Youcef Bey, Kyo Kageura, Christian Boitet et Francesca Marzari (2008)** *Translating the DEMGOL Etymological Dictionary of Greek Mythology with the BEYTrans Wiki*. In the Proceeding of the 4th International Symposium on Wikis, Wikisym2008. Porto, Portugal, 8-10 September, 2008, p. 9.

[Blanc, 1999a] **Étienne Blanc (1999a)** *An interactive hypertextual environment for MT development*. Journal of Chinese Language and Computing, Communication of COLIPS (Chinese and Oriental Languages Information Processing Society), Volume 9, Issue 1, pp. 67-81.

[Blanc, 1999b] **Étienne Blanc (1999b)** *PARAX-UNL: a large scale hypertextual multilingual lexical database*. NLPRS '99: the 5th Natural Language Processing Pacific Rim Symposium, Beijing, China, 4 p.

[Blanc, 2000] **Étienne Blanc (2000)** *From the UNL hypergraph to GETA's multilevel tree*. Proc. MT'2000, Oxford, 18-21 Oct. 2000, British Computer Society, 10 p.

[Blanchon et Boitet, 1994] **Hervé Blanchon et Christian Boitet (1994)** *Multilingual dialogue-based MT for monolingual authors : the LIDIA Project and a first mockup*. Machine translation ISSN 0922-6567, vol. 9, no. 2, pp. 99-132.

[Blanchon et Boitet, 2004] **Hervé Blanchon et Christian Boitet (2004)** *Deux premières étapes vers les documents auto-explicatifs*. Proc. TALN 2004, Fès, Maroc, 19-21 avril 2004, vol. 1/1, pp. 61-70.

[Blanchon et Boitet, 2007] **Hervé Blanchon et Christian Boitet (2007)** *Projet TRANSAT: État de l'art en traduction de l'écrit*. Rapport actualisé, p. 68.

[Blanchon, Boitet et Huynh, 2009] **Hervé Blanchon, Christian Boitet et Cong-Phap Huynh (2009)** *A Web Service Enabling Gradable Post-edition of Pre-translations Produced by Existing Translation Tools: Practical Use to Provide High-quality Translation of an Online Encyclopedia*. MT Summit XII, Ottawa, Ontario, Canada, August 26-30, 2009, 8 p.

[Bläter, Schwall et Storrer, 1992] **Brigitte Bläter, Ulrikf Schwall et Angelika Storrer (1992)** *A Reusable Lexical Database Tool For Macitlne Translation*. Proc. of COLING-92, Nantes, Aug. 23-98, 1992, pp. 510-517.

[Boitet, 1986] **Christian Boitet (1986)** *The French National Project: technical organization and translation results of CALLIOPE-AERO*. IBM Conference on machine translation, Copenhagen, August 1986, pp. 239-267.

[Boitet, 1990] **Christian Boitet (1990)** *Towards Personal MT: general design, dialogue structure, potential role of speech*. Proc. COLING-90, Helsinki, Finland, August 20-25, 1990, vol. 3/3, pp. 30-35.

[Boitet, 1993a] **Christian Boitet (1993a)** *Crucial open problems in Machine Translation & Interpretation*. Proc. Symposium on Natural Language Processing in Thailand, Bangkok, 17-20 March 1993, pp. 1-29.

[Boitet, 1993b] **Christian Boitet (1993b)** *TA et TAO à Grenoble... 32 ans déjà ! T.A.L (revue semestrielle de l'ATALA)*, vol.33/1-2, Spécial Trentenaire, pp. 45-84.

[Boitet, 1996] **Christian Boitet (1996)** *La synergie entre THAM, réseau et TA comme facteur de progrès théoriques et pratique en TAO*. NLP+IA 96 / TAL+AI 96, Université de Moncton, Canada, 12 p.

[Boitet, 1999] **Christian Boitet (1999)** *A research perspective on how to democratize machine translation and translation aids aiming at high quality final output*. MT Summit VII, September 13-17, 1999, Kent Ridge Digital Labs, Singapore, pp. 125-133.

[Boitet, 2002] **Christian Boitet (2002)** *A roadmap for MT : four « keys » to handle more languages, for all kinds of tasks, while making it possible to improve quality (on demand)*. International Conference on Universal Knowledge and Language (ICUKL2002), Goa, 25-29 November 2002, 8 p.

[Boitet, 2005] **Christian Boitet (2005)** *What will it take for Papillon to be concretely useful not only for humans, but for machines ?* Proc. SNLP-2005, Bangkok, Thailand, 10 p.

[Boitet, 2007] **Christian Boitet (2007)** *Corpus pour la TA : types, tailles et problèmes associés, selon leur usage et le type de système*. Revue française de linguistique appliquée Pub. linguistiques, Vol. XII 2007/1, pp. 25-38.

[Boitet, 2008] **Christian Boitet (2008)** *Les architectures linguistiques et computationnelles en traduction automatique sont indépendantes*. JEP-TALN 2008, Avignon, 9-13 juin 2008, 10 p.

[Boitet, Boguslavskij et Cardeñosa, 2007] **Christian Boitet, Igor Boguslavskij et Igor Cardeñosa (2007)** *An Evaluation of UNL Usability for High Quality Multilingualization and Projections for a Future UNL++ Language*. Proc. of Computational Linguistics and Intelligent Text Processing (CICLING-2007), February 18 to 24, 2007, Mexico City, Mexico, pp. 361-373.

[Boitet, Chatelin et Fraga, 1980] **Christian Boitet, Philippe Chatelin et Paltonio Daun Fraga (1980)** *Present and future paradigms in the automatized translation of natural languages*. International Conference On Computational Linguistics, Proceedings of the 8th conference on Computational linguistics, Tokyo, Japan, SESSION: Informatics: machine translation, pp. 430-436.

[Boitet, Guillaume et Quézel-Ambrunaz, 1978] **Christian Boitet, Pierre Guillaume et Maurice Quézel-Ambrunaz (1978)** *Manipulation d'arborescences et parallélisme: le système ROBRA*. Proceedings of COLING- 78, Bergen Norway, August, 1978, 12 p.

[Boitet et al., 2008] **Christian Boitet, Cong-Phap Huynh, Hervé Blanchon, Hong-Thai Nguyen et David Rouquet (2008)** *A Web-oriented System to Manage the Translation of an Online Encyclopedia Using Classical MT and Deconversion from UNL*. ASWC/IC-08, Bangkok, 11 p.

[Boitet, Mangeot et Serasset, 2002] **Christian Boitet, Mathieu Mangeot et Gilles Serasset (2002)** *The Papillon project: cooperatively building a multilingual lexical data-base to derive open source dictionaries and lexicons*. Proceedings of the 2nd workshop on NLP and XML, volume 17, pp. 1-3.

[Boitet et Nedobejkine, 1980] **Christian Boitet et Nicolas Nedobejkine (1980)** *Russian-French at GETA: outline of the method and detailed example*. Proceedings of the 8th conference on Computational linguistics, COLING 80, Tokyo, Japan, pp. 437-446.

[Boitet et Nedobejkine, 1986] **Christian Boitet et Nicolas Nedobejkine (1986)** *Towards integrated dictionaries for M(a)T: motivations and linguistic organization*. Proc. COLING-86, Vol. 1, 25-29 août 1986, Bonn, pp. 423-428.

[Boitet et Seligman, 1994] **Christian Boitet et Mark Seligman (1994)** *The "Whiteboard" architecture: a way to integrate heterogeneous components of NLP systems*. Proc. COLING-94, vol. 1/2, Kyoto, Japan, August 5-9, 1994, pp. 426-430.

[Boitet et Zaharin, 1988] **Christian Boitet et Yusoff Zaharin (1988)** *Representation trees and string-tree correspondences*. Proc. COLING-88, 12th conference on Computational linguistics, Volume 1, Budapest, Hungary, 1988, pp. 59-64.

[Bouchou et Maurel, 1999] **Béatrice Bouchou et Denis Maurel (1999)** *Using Transducers in Natural Language Database Query System*. Third International Workshop on Applications of Natural Language to Data Bases (NLDB'99), Klagenfurt, Autriche, 17-19 juin 1999, pp. 17-31.

[Bouillon, Boesefeldt et Russell, 1992] **Pierrette Bouillon, Katharina Boesefeldt et Graham Russell (1992)** *Compound Nouns in a Unification-Based MT System*. Proc. of the third conference on Applied natural language processing, Trento, Italy, pp. 209-215.

[Brants et Plaehn, 2000] **Thorsten Brants et Oliver Plaehn (2000)** *Interactive Corpus Annotation*. Second International Conference on Language Resources and Evaluation LREC-2000, May 31 – June 2, 2000, Athens, Grece, 7 p.

[Brown et al., 1990] **Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer et Paul S. Roossin (1990)** *A Statistical Approach to Machine Translation*. Computational Linguistics, Volume 16, Number 2, June 1990, pp. 79-85.

[Brown et al., 1993] **Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra et Robert L. Mercer (1993)** *The Mathematics of Statistical Machine Translation: Parameter Estimation*. Computational Linguistics, Volume 19, Number 2, 1993, pp. 263-331.

[Brown, 1996] **Ralf D. Brown (1996)** *Example-Based Machine Translation in the Pangloss System*. Proceedings of the 16th conference on Computational linguistics - Volume 1, Copenhagen, Denmark, pp. 169-174.

[Cabré, Sager et DeCesaris, 1999] **Maria Teresa Cabré, Juan C. Sager et Janet Ann DeCesaris (1999)** *Terminology: theory, methods, and applications*. John Benjamins Publishing Company Ed., 1999. 247 p.

[Calzolari et al., 2001] **Nicoletta Calzolari, Alessandro Lenci, Antonio Zampolli, Nuria Bel, Marta Villegas et Gregor Thurmair (2001)** *The ISLE in the ocean, transatlantic*

standards for multilingual lexicons (with an eye to machine translation). MT Summit VIII, Machine Translation in the Information Age, Santiago de Compostela, Spain, 18-22 September 2001, 5 p.

[Cardeñosa, Gallardo et Iraola, 2004] **Jesús Cardeñosa, Carolina Gallardo et Luis Iraola (2004)** *The forgotten key point for assuring knowledge consistency in CLIR systems*. Proceedings Workshop Lessons Learned from Evaluation: Towards Transparency & Integration in Cross-Lingual Information Retrieval. (LREC, 2004), Lisboa, Mayo 2004, 13 p.

[Cardeñosa et al., 2004] **Jesús Cardeñosa, Carolina Gallardo, Luis Iraola et Edmundo Tovar (2004)** *A Multilingual Approach for Web Applications: The UNL System*. Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics. Vol. 10, Orlando (Florida), June 2004, pp. 203-207.

[Cardeñosa, Gallardo et Santos, 2005] **Jesús Cardeñosa, Carolina Gallardo et Eugenio Santos (2005)** *Bridging the gap between human language and computer-oriented representations*. International Journal "Information Theories and Applications". Vol 12. ISSN: 1310-0513.

[Cardeñosa, Gallardo et Tova, 2005] **Jesús Cardeñosa, Carolina Gallardo et Edmundo Tova (2005)** *Towards a systematic process in the use of UNL to support multilingual services*. Research on Computing Science, Vol. 12. ISSN: 1665-9899, 17 p.

[Carl, 2000] **Michael Carl (2000)** *Combining Invertible Example-Based Machine Translation with Translation Memory Technology*. Proceedings of the 4th Conference of the Association for Machine Translation in the Americas on Envisioning Machine Translation in the Information Future, pp. 127-136.

[Carl, Way et Schäler, 2002] **Michael Carl, Andy Way et Reinhard Schäler (2002)** *Toward a Hybrid Integrated Translation Environment*. Proceedings of AMTA (The Association for Machine Translation in the America), October 8-12, 2002, Tiburon, California, pp. 11-21.

[Carpena, 2004] **Virginie Carpena (2004)** *WICALE, une interface cliente générique pour le pilotage de serveurs linguistiques*. Mémoire d'ingénieur CNAM, GETA, CLIPS, IMAG, 86 p.

[Castelli, 2004] **Éric Castelli (2004)** *Natural Language Processing & Speech Processing*. The second seminar of the ICT-Asia Programme. Kuala Lumpur, Malaysia, November 29 - December 1st, 2004.

[Castelli et Nguyen, 2001] **Éric Castelli et Trong-Giang Nguyen (2001)** *Le Centre de Recherche International MICA. Une nouvelle étape dans la coopération universitaire franco-vietnamienne*. Actes de la 19eme Conférence scientifique internationale, Session : Nouvelles technologies pour l'Éducation, Institut Polytechnique de Hanoi, octobre 2001, pp. 9-13.

[Chalvin et Mangeot, 2006] **Antoine Chalvin et Mathieu Mangeot (2006)** *Méthodes et outils pour la lexicographie bilingue en ligne : le cas du grand dictionnaire estonien-français (GDEF)*. Proceedings XII Euralex International Congress, Torino, Italy, September 6th-9th, 2006, Alessandria : Edizioni dell'Orso, 2006, vol. I, pp. 605-610.

[Chandioux et Guéraud, 1981] **John Chandioux et Marie-France Guéraud (1981)** *METEO: un système à l'épreuve du temps*. META 26(1), pp. 17-22.

[Chang, Luo et Su, 1992] **Jing-Shin Chang, Yih-Fen Luo et Keh-Yih Su (1992)** *GPSM: a Generalized Probabilistic Semantic Model for Ambiguity Resolution*. Proceedings of ACL-92, 30th Annual Meeting of the Association for Computational Linguistics, University of Delaware, Newark, DE, USA, 28 June-2 July, 1992, pp. 177-184.

[Chappuy, 1993] **Sylviane Chappuy (1993)** *Formalisation de la description des niveaux d'interprétation des langues naturelles. Etudes menées en vue de l'analyse et de la génération au moyen de transducteurs*. Thèse INPG, Grenoble, 2 juillet 1993, 213 p.

- [Charras et Lecroq, 2004] **Christian Charras et Thierry Lecroq (2004)** *Handbook of Exact String Matching Algorithms*. King's College Publications, ISBN: 0954300645. 256 p.
- [Chauché, 1975] **Jacques Chauché (1975)** *Les langages ATEF et CETA*. AJCL, microfiche 17, pp. 21-39.
- [Chauché, 1984] **Jacques Chauché (1984)** *Un outil multidimensionnel de l'analyse du discours*. Proceedings COLING-84, 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics (ACL), Stanford, California, pp. 11-15.
- [Chauché et al., 2003] **Jacques Chauché, Violaine Prince, Simon Jaillet et Maguelonne Teisseire (2003)** *Classification automatique de textes à partir de leur analyse syntaxico-sémantique*. TALN 2003, Batz-sur-Mer, 11-14 juin 2003, pp. 55-65.
- [Chenon, 2005] **Christophe Chenon (2005)** *Vers une meilleure utilisabilité des mémoires de traductions, fondée sur un alignement sous-phrastique*. Thèse UJF (Grenoble 1), 28 octobre 2005, 228 p.
- [Chenon, 2006] **Christophe Chenon (2006)** *TransTree, a formalism to capture nested correspondences at sub-sentential level*. International Symposium on Parallel Treebanks, Stockholm, 21-22 September 2006, 8 p.
- [Cicekli et Guvenir, 2001] **Ilyas Cicekli et H. Altay Guvenir (2001)** *Learning Translation Templates from Bilingual Translation Examples*. Applied Intelligence, Vol 15., No. 1, 2001, pp. 57-76.
- [Collins et Cunningham, 1996a] **Bróna Collins et Padraig Cunningham (1996a)** *A Case-Based Approach to Machine Translation*. Advances in Case-Based Reasoning, Third European Workshop, EWCBR-96, November 14-16, 1996, Lausanne, Switzerland pp. 91-104.
- [Collins et Cunningham, 1996b] **Bróna Collins et Padraig Cunningham (1996b)** *Adaptation Guided Retrieval in EBMT: A Case-Based Approach to Machine Translation*. Springer-Verlag London, UK (Ed.) Proceedings of the Third European Workshop on Advances in Case-Based Reasoning, Lecture Notes In Computer Science; Vol. 1168, pp. 91-104.
- [Collins et Somers, 2003] **Bróna Collins et Harold Somers (2003)** *EBMT seen as case-based reasoning*. n Michael Carl and Andy Way (eds) Recent advances in Example-Based Machine Translation, Dordrecht: Kluwer, pp. 115-153.
- [Colmerauer, 1967] **Alain Colmerauer (1967)** *Les SYSTEMES-Q: un formalisme pour analyser et synthétiser des phrases sur ordinateur*. Groupe TAUM, Université de Montréal, Montréal, Canada, 28 p.
- [Costa et Panissod, 2003] **Jean-Cédric Costa et Christiane Panissod (2003)** *SYSTRAN Review Manager*. New Orleans, USA, 23-27 September 2003, pp. 451-454.
- [Daoud, Daoud et Boitet, 2009] **Mohammad Daoud, Daoud Daoud et Christian Boitet (2009)** *Collaborative Construction of Arabic Lexical Resources*. Khalid Choukri and Bente Maegaard (Ed.): Proceedings of the Second International Conference on Arabic Language Resources and Tools, April 22-23 2009, Cairo, Egypt, pp. 119-124.
- [DellaPietra et DellaPietra, 1994] **Stephen DellaPietra et Vincent DellaPietra (1994)** *Candide: A Statistical Machine Translation System*. Proceedings of the workshop on Human Language Technology, HLT-94, Plainsboro, NJ, pp. 457-457.
- [Désilets et al., 2006] **Alain Désilets, Lucas Gonzalez, Sébastien Paquet et Marta Stojanovic (2006)** *Translation the Wiki Way*. Proceedings of the WIKISym 2006 - The Conference Wiki of the 2006 International Symposium. Odense, Denmark. August 21-23, 2006 pp. 19-31.

- [Dillinger, 2001] **Mike Dillinger (2001)** *Dictionary Development Workflow for MT: Design and Management*. MT summit VIII, Santiago de Compostela, Espagne (18/09/2001), 2001, pp. 83-87.
- [Durand, 1988] **Jean-Claude Durand (1988)** *TTEDIT: un éditeur transformationnel d'arbres décorés*. Thèse UJF (Grenoble 1), mars 1988, 120 p.
- [Eberle, 2001] **Kurt Eberle (2001)** *Translation mismatches in lexically-driven FUDR-based MT*. TALN 2001, Tours, 2-5 juillet 2001, pp. 267-276.
- [Erl, 2005] **Thomas Erl (2005)** *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall PTR (August 12, 2005), ISBN-10: 0131858580. 792 p.
- [Etzioni et al., 2007] **Oren Etzioni, Kobi Reiter, Stephen Soderland et Marcus Sammer (2007)** *Lexical translation with application to image searching on the web*. MT Summit XI, Copenhagen, Denmark, September, 2007, pp. 175-182.
- [Frederking et al., 1994] **Robert Frederking, Sergei Nirenburg, D. Farwell, S. Helmreich, E.H. Hovy, K. Knight, S. Beale, C. Domashnev, D. Attardo, D. Grannes et Ralf D. Brown (1994)** *The Pangloss Mark III Machine Translation System*. Proceedings of the First Conference of the Association for Machine Translation in the Americas (AMTA-94). Columbia, Maryland 1994, 10 p.
- [Funaki, 1993] **Susumu Funaki (1993)** *Multilingual Machine Translation (MMT) Project*. MT Summit IV, July 20-22, 1993, Kobe, Japan, pp. 73-78.
- [Gaiffe, Jacquy et Kister, 2009] **Bertrand Gaiffe, Evelyne Jacquy et Laurence Kister (2009)** *Approche lexico-sémantique de l'extraction terminologique : utilisation de ressources lexicographiques et validation sur corpus*. Terminologie & Ontologie: Théories et Applications (TOTh-09), Annecy, 4-5 juin 2009, 10 p.
- [Gao et al., 2006] **Yuqing Gao, Liang Gu, Bowen Zhou, Ruhi Sarikaya, Mohamed Afify, Hong-Kwang Kuo, Wei-Zhong Zhu, Yonggang Deng, Charles Prosser, Wei Zhang et Laurent Besacier (2006)** *IBM Mastor: Multilingual Automatic Speech-To-Speech Translator*. Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. IEEE International Conference, May 14-19, 2006, Toulouse, France, 5 p.
- [Gaschler et Lafourcade, 1994] **Jean Gaschler et Mathieu Lafourcade (1994)** *Manipulating human-oriented dictionaries with very simple tools*. Proceedings of the 15th conference on Computational linguistics, COLING-94, Kyoto, Japan, Volume 1, pp. 283-286.
- [Genthial, 1991] **Damiel Genthial (1991)** *Contribution à la construction d'un système robuste d'analyse du français*. Thèse UJF (Grenoble 1), 10 janvier 1991, 248 p.
- [Gerber, 1984] **René Gerber (1984)** *Etude des possibilités de coopération entre un système fondé sur des techniques de compréhension implicite (système logico-syntaxique) et un système fondé sur des techniques de compréhension explicite (système expert)*, Thèse USTMG, Grenoble, 115 p.
- [Ginon, 2007] **Yohan Ginon (2007)** *Import de données lexicales pour le projet PIVAX*. Rapport de TER d'informatique, UJF, LIG, GETALP, 9 p.
- [Guilbaud, 1988] **Jean-Philippe Guilbaud (1988)** *Projet Eurotra: Réalisation en Ariane d'un transfert des structures produites par l'analyseur du français de B'VITAL vers des structures interface IS Eurotra*. Rapport de contrat, GETA, juin 1988, 205 p.
- [Guillaume, 1989a] **Pierre Guillaume (1989a)** *Ariane-G5 - Extensions apportées au langage SYGMOR*. Document interne GETA, janvier 1989 (Ariane-G5 version 3), 6 p.
- [Guillaume, 1989b] **Pierre Guillaume (1989b)** *Ariane-G5 - Les langages spécialisés TRACOMPL et EXPANS*. Document interne GETA, septembre 1989 (Ariane-G5 version 3), 93 p.

- [Guillaume, 2003] **Pierre Guillaume (2003)** *Le réseau LIDIA*. Document interne GETA, octobre 2003, 5 p.
- [Habash, 2002] **Nizar Habash (2002)** *Generation-heavy hybrid machine translation*. Proceedings of the International Natural Language Generation Conference (INLG'02), New York, pp. 61-69.
- [Hermjakob et Mooney, 1997] **Ulf Hermjakob et Raymond J. Mooney (1997)** *Learning Parse and Translation Decisions From Examples With Rich Context*. Proceedings of the Conference of the Association for Computational Linguistics (ACL/EACL), Madrid, Spain, 1997, pp. 482-489.
- [Hofmann et Harris, 1970] **Th. R. Hofmann et Brian Harris (1970)** *Pidgin Translation*. Meta, volume 15, number 2, 1970, pp. 71-87.
- [Huang et al., 2008] **Liang Huang, Hao Zhang, Daniel Gildea et Kevin Knight (2008)** *Binarization of Synchronous Context-Free Grammars*. Proc. COLING-2008, Manchester, 18-22 August, 2008, 32 p.
- [Hutchins, 1978] **W. John Hutchins (1978)** *Machine Translation and Machine-Aided Translation*. Journal of Documentation, Vol.34, No.2, June 1978, pp. 119-159.
- [Hutchins, 1981] **W. John Hutchins (1981)** *The Evolution of Machine Translation Systems*. Proceedings of Practical experience of Machine Translation, London, 5-6 November 1981, pp. 21-37.
- [Hutchins, 1999] **W. John Hutchins (1999)** *The development and use of machine translation systems and computer-based translation tools*. International Symposium on Machine Translation and Computer Language Information Processing, 26-28 June 1999, Beijing, China, pp. 1-16.
- [Hutchins et Somers, 1986] **W. John Hutchins et H. L. Somers (1986)** *Machine Translation: Past, Present and Future*. Chichester (UK), Ellis Horwood, 1986, (ISBN: 0-85312-788-3). 382 p.
- [Huynh, Boitet et Blanchon, 2008] **Cong-Phap Huynh, Christian Boitet et Hervé Blanchon (2008)** *SECTra_w.1: an online collaborative system for evaluating, post-editing and presenting MT translation corpora*. LREC 2008: 6th Language Resources and Evaluation Conference, Marrakech, Morocco, 26-30 May 2008, 6 p.
- [Ibanescu, Buche et Dibia-Barthélemy, 2009] **Liliana Ibanescu, Patrice Buche et Juliette Dibia-Barthélemy (2009)** *Construction et alignement d'ontologies pour évaluer le risque alimentaire*. Terminologie & Ontologie: Théories et Applications (TOTh-09), Annecy, 4-5 juin 2009, 10 p.
- [Ikehara et al., 1991] **Satoru Ikehara, Satoshi Shirai, Akio Yokoo et Hiromi Nakaiwa (1991)** *Toward an MT system without pre-editing effects of new method in ALT-J/E*. MT Summit III, July 1-4, 1991, Washinton DC, pp. 101-106.
- [Imamura, 2001] **Kenji Imamura (2001)** *Hierarchical Phrase Alignment Harmonized with Parsing*. Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium, November 27-30, 2001, Hitotsubashi Memorial Hall, National Center of Sciences, Tokyo, Japan, pp. 377-384.
- [Iraola, 2005] **Luis Iraola (2005)** *Using WordNet for linking UWs to the UNL UW system*. Jesús Cardeñosa, Alexander Gelbukh, Edmundo Tovar (eds.): Universal Networking Language: advances in theory and applications, National Polytechnic Institute, Mexico City, pp. 370-379.
- [Isabelle et Bourbeau, 1984] **Pierre Isabelle et Laurent Bourbeau (1984)** *TAUM-AVIATION: its technical features and some experimental results*. Comp. Ling., 11/1, pp. 18-27.

- [Jones et al., 2008] **Gareth J. F. Jones, Fabio Fantino, Eamonn Newman et Ying Zhang (2008)** *Domain-specific query translation for multilingual information access using machine translation augmented with dictionaries mined from Wikipedia*. Workshop in conjunction with IJCNLP 2008 – The Third International Joint Conference on Natural Language Processing, 7-12 Jan., 2008, Hyderabad, India, pp. 34-41.
- [Kakkonen, 2005] **Tuomo Kakkonen (2005)** *DepAnn-An Annotation Tool for Dependency Treebanks*. Proc. of the 11th ESSLLI Student Session, Malaga, pp. 214-225.
- [Kamei, 1999] **Shin-ichiro Kamei (1999)** *Sharing Dictionaries among MT Users By Common Formats and Social Filtering Framework*. MT Summit VII, September 13-17, 1999, Kent Ridge Digital Labs, Singapore, pp. 180-181.
- [Kamei et al., 1997] **Shin-Ichiro Kamei, Etsuo Itoh, Mikiko Fujii, Tokuyuki Hirai, Yukari Saitoh, Masahito Takahashi, Tsutomu Hiyama et Kazunori Muraki (1997)** *Sharable Formats and their Supporting Environments for Exchanging User Dictionaries among Different MT Systems As a part of AAMT Activities*. MT Summit VI, San Diego, 29 October-1 November 1997, pp. 132-141.
- [Kay, 1984] **Martin Kay (1984)** *Functional unification grammar: a formalism for machine translation*. Proceedings of the 10th International Conference on Computational Linguistics, Stanford University, California, July 1984, ACL, pp. 75-78.
- [King et Falkedal, 1990] **Margaret King et Kirsten Falkedal (1990)** *Using test suites in evaluation of machine translation systems*. Proceedings of the 13th International Conference on Computational linguistics (COLING-90), Volume 2, Helsinki, Finland, pp. 211-216.
- [Koehn, 2004] **Philipp Koehn (2004)** *Pharaoh: A Beam Search Decoder for Phrase-Based Statistical Machine Translation Models*. Machine Translation: From Real Users to Research, 6th Conference of the Association for Machine Translation in the Americas, AMTA 2004, Washington, DC, USA, September 28-October 2, 2004, pp. 115-124.
- [L'haire, Mengon et Laenzlinger, 2000] **Sébastien L'haire, Juri Mengon et Christopher Laenzlinger (2000)** *Outils génériques et transfert hybride pour la traduction automatique sur Internet*. Conférence TALN 2000, Lausanne, 16-18 octobre 2000, 10 p.
- [Lafourcade, 1994] **Mathieu Lafourcade (1994)** *Génie logiciel pour le génie linguiciel*. Thèse UJF (Grenoble 1), 308 p.
- [Lafourcade, 1997] **Mathieu Lafourcade (1997)** *Multilingual dictionary construction and services case study with the Fe* projects*. PACLING'97 - September 2/5 1997 - Meisei University - Ohme, Tokyo, Japan, pp. 171-181.
- [Lafourcade, Prince et Schwab, 2002] **Mathieu Lafourcade, Violaine Prince et Didier Schwab (2002)** *Vecteurs conceptuels et structuration émergente de terminologies*. TAL, 43(1), pp. 43-72.
- [Lau, 1987] **Peter Lau (1987)** *Eurotra: past, present and future*. In proceedings of the International Conference Translating and the Computer, ASLIB 1987, 12-13 November 1987, CBI Conference Centre, Centre Point, London, pp. 186-191.
- [Lebarbé, 2007] **Thomas Lebarbé (2007)** *LexTract : extraction récursive de termes à portée juridique*. 5e Journées de Linguistique de Corpus, Lorient, 13 - 15 septembre 2007, pp. 197-205.
- [Leon, 2000] **Marjorie Leon (2000)** *A new look for the PAHO MT system*. AMTA 2000 : Association for Machine Translation in the Americas. Conference No 4, Cuernavaca, Mexico, October 10, 2000, vol. 1934, ISBN 3-540-41117-8, pp. 219-222.
- [León, 2001] **Marjorie León (2001)** *SPANAM® and ENGSPAN® for Windows 2000: an MT pioneer keeps up with technology*. MT Summit VIII, Machine Translation in the Information Age, Santiago de Compostela, Spain, 18-22 September 2001, pp. 203-206.

- [Lepage, 1986] **Yves Lepage (1986)** *TL, a language for transcriptions*. Proc. COLING-86, Bonn, August 25-29, 1986, University of Bonn, Germany, pp. 402-404.
- [Lepage, 1989] **Yves Lepage (1989)** *Un système de grammaires correspondancier les d'identification*. Thèse UJF (Grenoble 1), 181 p.
- [Lepage, 1997] **Yves Lepage (1997)** *String Approximate Pattern-Matching*. 55th Meeting of the Information Processing Society of Japan, Fukuoka, August 1997, vol. 3, pp. 139-140.
- [Lepage, 2003] **Yves Lepage (2003)** *De l'analogie rendant compte de la commutation en linguistique*. Mémoire de HDR, UJF (Grenoble 1), 388 p.
- [Lepage et Denoual, 2005] **Yves Lepage et Etienne Denoual (2005)** *The purest EBMT system ever built: no variables, no templates, no training, examples, just examples, only examples*. Proceedings of the Workshop on Example-based Machine Translation, hosted by MT Summit X (Phuket, Thailand, Sept. 2005), pp. 81-90.
- [Lieske, McCormick et Thurmair, 2001] **Christian Lieske, Susan McCormick et Gregor Thurmair (2001)** *The Open Lexicon Interchange Format (OLIF) comes of age*. MT Summit VIII, Machine Translation in the Information Age, Santiago de Compostela, Spain, 18-22 September 2001, pp. 211-216.
- [Mangeot, 2001] **Mathieu Mangeot (2001)** *Environnements centralisés et distribués pour lexicographes et lexicologues en contexte multilingue*. Thèse UJF (Grenoble 1), 296 p.
- [Mangeot, 2002] **Mathieu Mangeot (2002)** *How to Import an Existing XML Dictionary Into the Papillon Platform*. Proceedings of Papillon 2002 Workshop, 16-18 July 2002, NII, Tokyo, Japan, 10 p.
- [Mangeot et Nguyen, 2009] **Mathieu Mangeot et Hong-Thai Nguyen (2009)** *Building lexical resources: towards programmable contributive platforms*. IEEE-RIVF 2009, International Conference on Computing and Communication Technologies, July 13-17, 2009. Danang, Vietnam, pp. 84-92.
- [Maurel et Guenther, 2006] **Denis Maurel et Franz Guenther (2006)** *Automata and Dictionaries*. Texts in computing, vol. 6, King's College Publications, London, ISBN 1-904-987-32-X. 240 p.
- [McCord, 1989] **Michael C. McCord (1989)** *Design of LMT: a Prolog-based Machine Translation System*. Computational Linguistics, Volume 15, Number 1, March 1989, pp. 33-52.
- [Melby, 1982] **Alan K. Melby (1982)** *Multi-level translation aids in a distributed system*. Proceedings of the 9th International Conference on Computational Linguistics (COLING-82), Volume 1, Prague, Czechoslovakia, Ed. Academia Praha, pp. 215-220.
- [Melby, 1983] **Alan K. Melby (1983)** *Computer-assisted translation systems: the standard design and a multi-level design*. Applied Natural Language Conferences, Proceedings of the first conference on Applied natural language processing, Santa Monica, California, pp. 174-177.
- [Melby, 1998] **Alan K. Melby (1998)** *Some Notes on The Proper Place of Men and Machines in Language Translation*. Machine Translation, Volume 12, Issue 1-2, pp. 29-34.
- [Melby, Smith et Peterson, 1980] **Alan K. Melby, Melvin R. Smith et Jill Peterson (1980)** *ITS: interactive translation system*. Proceedings of the 8th International Conference on Computational Linguistics (COLING-80), Tokyo, Japan, pp. 424-429.
- [Mitamura et Nyberg, 1992a] **Teruko Mitamura et Eric Nyberg (1992a)** *Hierarchical Lexical Structure and Interpretive Mapping in Machine Translation*. Proceedings of the 15th International Conference on Computational Linguistics (COLING-92), Volume 4, Nantes, France, pp. 1254-1258.

[Mitamura et Nyberg, 1992b] **Teruko Mitamura et Eric Nyberg (1992b)** *The KANT System: Fast, Accurate, High-Quality Translation in Practical Domains*. Proceedings of the 15th International Conference on Computational Linguistics (COLING-92), Nantes, France, p. 1069-1073.

[Mitamura et Nyberg, 2000] **Teruko Mitamura et Eric Nyberg (2000)** *The KANTOO Machine Translation Environment*. Proceedings of the 4th Conference of the Association for Machine Translation in the Americas on Envisioning Machine Translation in the Information Future (AMTA 2000), October 2000, Cuernavaca, Mexico, pp. 192-195.

[Murata et al., 2003] **Toshiki Murata, Mihoko Kitamura, Tsuyoshi Fukui et Tatsuya Sukehiro (2003)** *Implementation of Collaborative Translation Environment 'Yakushite Net'*. Machine Translation Summit IX, September 23-27, 2003, New Orleans, Louisiana, USA, pp. 479-482.

[Nagao, 1984] **Makoto Nagao (1984)** *A framework of a mechanical translation between Japanese and English by analogy principle*. Proc. of the international NATO symposium on artificial and human intelligence, Lyon, France, ISBN: 0-444-86545-4, pp. 173-180.

[Nagao et Tsujii, 1986] **Makoto Nagao et Jun-ichi Tsujii (1986)** *The Transfer Phase of MU Machine Translation System*. Proceedings of the 11th International Conference on Computational Linguistics COLING-86, Vol. 1, Bonn, Germany, pp. 97-103.

[Navarro et Raffinot, 2008] **Gonzalo Navarro et Mathieu Raffinot (2008)** *Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press; 1 edition (August 21, 2008), ISBN-10: 0521039932. 232 p.

[Nguyen, 1998] **Hai-Doan Nguyen (1998)** *Techniques génériques d'accumulation d'ensembles lexicaux structurés à partir de ressources dictionnairiques informatisées multilingues hétérogènes*. Thèse INPG, Grenoble, 22 décembre 1998, 184 p.

[Nguyen, 2005] **Hong-Thai Nguyen (2005)** *Vers un "méta-EDL", puis un "EDL générique" pour la TAO*. Mémoire de M2R, UJF (Grenoble 1), 85 p.

[Nguyen et Boitet, 2007] **Hong-Thai Nguyen et Christian Boitet (2007)** *Vers un méta-EDL complet, puis un EDL universel pour la TAO*. TALN 2007, Toulouse, 5-8 juin 2007, 10 p.

[Nguyen et Boitet, 2009] **Hong-Thai Nguyen et Christian Boitet (2009)** *Lexical synergy between MT & Translator Aids: PIVAX, a generic online contributive lexical database platform*. International Conference "Machine Translation 25 Years On", Cranfield, England, November 21-22, 2009, 8 p.

[Nguyen, Boitet et Sérasset, 2007] **Hong-Thai Nguyen, Christian Boitet et Gilles Sérasset (2007)** *PIVAX, an online contributive lexical data base for heterogeneous MT systems using a lexical pivot*. SNLP-2007, December 2007, Bangkok, Thailand, 6 p.

[Och et Ney, 2003] **Franz Josef Och et Hermann Ney (2003)** *A Systematic Comparison of Various Statistical Alignment Models*. Computational Linguistics, volume 29, number 1, March 2003, pp. 19-51.

[Ono et al., 2008] **Kinji Ono, Asanobu Kitamoto, Makiko Onishi, Elham Andaroodi, Yoko Nishimura et Mohammad Reza Matini (2008)** *Memory of the Silk Road -The Digital Silk Road Project-*. Proceedings of the Conference on Virtual Systems and Multimedia (VSM08), October 20. - 26th, 2008, Cyprus, Vol. Project Papers, pp. 437-444.

[Pankowicz, 1973] **Zbigniew B. Pankowicz (1973)** *User's evaluation of machine translation. Georgetown MT system (1963-1973)*. RADC-TR-73-239, Univ. of Texas at Austin.

[Pérez et al., 2006] **Carolina Gallardo Pérez, Jesús Cardeñosa, Igor Boguslavsky et Christian Boitet (2006)** *Summary of the U++ Consortium Meeting Minutes*. Paris, France, February 17th - 19th, 2006, Colegio de España, 7e bd Jourdan, 75014, Paris, 9 p.

- [Perscheid, 1985] **Margaret M. Perscheid (1985)** *Computer-Aided Translation at WCC*. CALIC Journal (The Computer Assisted Language Instruction Consortium), Vol 3, No. 1 (September 1985), pp. 22-24.
- [Planas, 1998] **Emmanuel Planas (1998)** *TELA : Structure et algorithmes pour la traduction fondée sur la mémoire*. Thèse UJF (Grenoble 1), 371 p.
- [Planas, 2005] **Emmanuel Planas (2005)** *SIMILIS: second-generation translation memory software*. In proceedings of the International Conference Translating and the Computer, ASLIB 2005, 27, 24-25 November 2005, London.
- [Polguère, 2004] **Alain Polguère (2004)** *Lexicologie et sémantique lexicale - Notions fondamentales*. Presses Université Montréal. 266 p.
- [Potet, 2009] **Marion Potet (2009)** *Méta-moteur de traduction automatique : proposition d'une métrique pour le classement de traductions*. Proc. RECITAL 2009. Senlis, France. 23-26 juin 2009. Vol. 1/1, pp. 373-382.
- [Queens et Recker-Hamm, 2005] **Frank Queens et Ute Recker-Hamm (2005)** *A Net-based Toolkit for Collaborative Editing and Publishing of Dictionaries*. Literary and Linguistic Computing Advance Access, Oxford Journal, Vol. 20: (Suppl 1), pp. 165-175.
- [Quézel-Ambrunaz, 1989] **Maurice Quézel-Ambrunaz (1989)** *Ariane-G5 Moniteur*. Documentation interne GETA, 85 p.
- [Ramisch, 2009] **Carlos Eduardo Ramisch (2009)** *Multi-word terminology extraction for domain-specific documents*. Mémoire de M2R G-INP, Grenoble, 77 p.
- [Rayner et al., 2008] **Manny Rayner, Pierrette Bouillon, Jane Brotanek, Glenn Flores, Sonia Halimi, Beth Ann Hockey, Hitoshi Isahara, Kyoko Kanzaki, Elisabeth Kron, Yukie Nakao, Marianne Santaholma, Marianne Starlander et Nikos Tsourakis (2008)** *The 2008 MedSLT System*. Coling 2008, Proceedings of the workshop on Speech Processing for Safety Critical Translation and Pervasive Applications, Manchester, August 2008, pp. 32-35.
- [Reinke, 1999] **Uwe Reinke (1999)** *Towards a Closer Integration of Termbases, Translation Memories, and Parallel Corpora – A Translation-Oriented View –*. Proceedings of the 5th International Congress on Terminology and Knowledge Engineering, 23-27 August 1999, Innsbruck, Austria, pp. 527-543.
- [Reutter et al., 1997] **Thomas Reutter, Michael Gamon, Carmen Lozano et Jessie Pinkham (1997)** *Practical Experience with Grammar Sharing in Multilingual NLP*. Proceedings of 26th Annual Meeting of ACL, 7-12 July 1997, Madrid, Spain, pp. 49-56.
- [Rico, 2000] **Celia Rico (2000)** *Evaluation Metrics for Translation Memories*. Language International, vol. 12, no. 6, December 2000, pp. 36-37.
- [Rossi, 1982] **Francesco Rossi (1982)** *"The Impact of Post-Editors" Feedback on the Quality of MT*. In Lawson (1986), pp. 113-118.
- [Rouquet et Nguyen, 2009a] **David Rouquet et Hong-Thai Nguyen (2009a)** *Interlingual annotation of texts in the OMNIA project* 4th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics, November 6-8, 2009, Poznań, Poland, 5 p.
- [Rouquet et Nguyen, 2009b] **David Rouquet et Hong-Thai Nguyen (2009b)** *Multilinguisation d'une ontologie par des correspondances avec un lexique pivot*. Terminologie & Ontologie: Théories et Applications (TOTh-09), Annecy, 4-5 juin 2009, 19 p.
- [Santaholma, 2007] **Marianne Santaholma (2007)** *Grammar Sharing Techniques for Rule-based Multilingual NLP Systems*. Proceedings of NODALIDA 2007, May 25-26, 2007, Tartu, Estonia, pp. 253-260.

[Satayamas, Boitet et Kawtrakul, 2007] **Vee Satayamas, Christian Boitet et Asanee Kawtrakul (2007)** *AnnotEd-w, a specialized editor for annotating word boundaries collaboratively*. The seventh international Symposium on Natural Language Processing (SNLP 2007), Pattaya, Chonburi, Thailand, December 13-15, 2007, pp. 137-143.

[Sato et Nakanishi, 1998] **Kengo Sato et Masakazu Nakanishi (1998)** *Maximum Entropy Model Learning of the Translation Rules*. Proceedings of the International Conference on Computational Linguistics (COLING/ACL-98), Montreal, Quebec, Canada, 1998, pp. 1171-1175.

[Schubert, 2002] **Klaus Schubert (2002)** *Esperanto as an intermediary language for machine translation*. In John Newton (ed.), *Computers in translation: a practical appraisal* (London: Routledge, 1992), pp. 78-95.

[Schwab et Lim, 2008] **Didier Schwab et Lian Tze Lim (2008)** *Blexisma2: a Distributed Agent Framework for Constructing a Semantic Lexical Database based on Conceptual Vectors*. DFMA-2008 : International Conference on Distributed Frameworks & Applications 2008, 21-22 October 2008, Penang, Malaysia, pp. 102-110.

[Scott, 1989] **Bernard E. Scott (1989)** *The Logos System*. MT Summit II: Final Programme, Exhibition, Papers, Munich, pp. 137-142.

[Senellart, Boitet et Romary, 2003] **Jean Senellart, Christian Boitet et Laurent Romary (2003)** *SYSTRAN new generation: the XML translation workflow*. MT Summit IX, New Orleans, USA, 23-27 September 2003, pp. 338-345.

[Senellart, Yang et Rebollo, 2003] **Jean Senellart, Jin Yang et Anabel Rebollo (2003)** *SYSTRAN Intuitive Coding Technology*. MT Summit IX, New Orleans, USA, September 22-26, 2003, pp. 346-353.

[Sérasset, 1994] **Gilles Sérasset (1994)** *SUBLIM : un système universel de bases lexicales multilingues et NADIA : sa spécialisation aux bases lexicales interlingues par acceptions*. Thèse UJF (Grenoble 1), 205 p.

[Sérasset, 1995] **Gilles Sérasset (1995)** *Interlingual Lexical Organisation for Multilingual Lexical Databases in NADIA*. COLING '94, 5-9 August 1994, Kyoto, Japan, pp. 278-282.

[Sérasset, 2004] **Gilles Sérasset (2004)** *A Generic Collaborative Platform for Multilingual Lexical Database Development*. COLING 2004, Workshop on Multilingual Linguistic Resources, Geneva, Switzerland, Aug. 2004, pp. 73-79.

[Sérasset, 2006] **Gilles Sérasset (2006)** *Multilingual legal terminology on the Jibiki platform: the LexALP project*. Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, Sydney, Australia, pp. 937-944.

[Sérasset et Blanc, 2003] **Gilles Sérasset et Étienne Blanc (2003)** *Remaining Issues that could Prevent UNL to be Accepted as a Standard*. Convergences'03, December 2 - 6, 2003, Alexandria, Egypt, pp. 117-124.

[Sérasset et Boitet, 1999] **Gilles Sérasset et Christian Boitet (1999)** *UNL-French deconversion as transfer & generation from an interlingua with possible quality enhancement through offline human interaction*. Proc. MT Summit VII, Singapore, 13-19 September 1999, Asia Pacific Ass. for MT, J.-I. Tsujii, pp. 220-228.

[Shimohata et al., 1999] **Sayori Shimohata, Toshiki Murata, Atsushi Ikeno, Tsuyoshi Fukui et Hideki Yamamoto (1999)** *Machine Translation System PENSÉE: System Design and Implementation*. Machine Translation Summit VII, 13th-17th September 1999, Kent Ridge Digital Labs, Singapore. Proceedings of MT Summit VII "MT in the Great Translation Era", pp. 380-384.

[Slocum, 1985] **Jonathan Slocum (1985)** *A Survey of Machine Translation: its History, Current Status, and Future Prospects*. Computational Linguistics, Volume 11, Number 1, January-March 1985.

[Sobashima et al., 1994] **Yasuhiro Sobashima, Osamu Furuse, Susumu Akamine, Jun Kawai et Hitoshi Iida (1994)** *A bidirectional, Transfer-Driven Machine Translation system for spoken dialogues*. Proceedings of the 15th conference on Computational linguistics - Volume 1, Kyoto, Japan, pp. 64-68.

[Stephen, 1994] **Graham A. Stephen (1994)** *String Searching Algorithms*. World Scientific Publishing Company; 1st edition (January 15, 1994). 243 p.

[Stewart, 1975] **Gilles Stewart (1975)** *Manuel du langage REZO*. TAUM, Univ. de Montréal, 120 p.

[Su et Chang, 1990] **Keh-Yih Su et Jing-Shin Chang (1990)** *Some key issues in designing MT systems*. Machine Translation 5(4), pp. 265-300.

[Su, Wu et Chang, 1992] **Keh-Yih Su, Ming-Wen Wu et Jing-Shin Chang (1992)** *A new quantitative quality measure for machine translation systems*. Proceedings of the 14th International Conference on Computational Linguistics (COLING-92), Volume 2, Nantes, France, pp. 433-439.

[Sumita et Iida, 1991] **Eiichiro Sumita et Hitoshi Iida (1991)** *Experiments and Prospects of Example-Based Machine Translation*. Proceedings. ACL 1991, 29th Annual Meeting of the Association for Computational Linguistics, 18-21 June 1991, University of California, Berkeley, California, USA, pp. 185-192.

[Tang, 1994] **Enya Kong Tang (1994)** *Natural Language Analysis In Machine Translation (MT) Based On The String-Tree Correspondence Grammar (STCG)*. PhD thesis, UTMK, Penang, Universiti Sains Malaysia, 253 p.

[Teeraparbserree, 2002] **Aree Teeraparbserree (2002)** *A Practical Guide to Lexical Data Acquisition with RECUPDIC*. Papillon 2002 Seminar, 16-18 July 2002, NII, Tokyo, Japan, 8 p.

[Teeraparbserree, 2005] **Aree Teeraparbserree (2005)** *Méthode et outils pour la création et l'évaluation automatiques de structures de bases lexicales multilingues (symétriques) à lexies et axes*. Thèse UJF (Grenoble 1), 166 p.

[Tesconi et al., 2006] **Maurizio Tesconi, Andrea Marchetti, Francesca Bertagna, Monica Monachini, Claudia Soria et Nicoletta Calzolari (2006)** *LeXFlow: A Framework for Cross-Fertilization of Computational Lexicons*. In Proceedings of COLING/ACL2006, Interactive Presentation Sessions, Sydney, Australia, 17th-21st July 2006, pp. 9-12.

[Thurmair, 2005] **Gregor Thurmair (2005)** *Hybrid Architectures for Machine Translation Systems*. Language Resources and Evaluation, Volume 39, Number 1, February, 2005, pp. 91-108.

[Tiedemann, 1999] **Jörg Tiedemann (1999)** *Uplug - a modular corpus tool for parallel corpora*. L. Borin (ed.) Parallel Corpora, Parallel Worlds. Proceedings of Parallel Corpus Symposium, Uppsala, April 22-23, 1999, 16 p.

[Tomokiyo, Mangeot et Planas, 2000] **Mutsuko Tomokiyo, Mathieu Mangeot et Emmanuel Planas (2000)** *Papillon : a Project of Lexical Database for English, French and Japanese, using Interlingual Links*. Journées des Sciences et Techniques de l'Ambassade de France au Japon, Tokyo, Japon, 13-14 novembre 2000, Ambassade de France au Japon, 3 p.

[Tong, 1990] **Loong Cheong Tong (1990)** *An explanation facility for a grammar writing system*. Proceedings COLING-90, 13th International Conference on Computational Linguistics - Volume 2, Helsinki, Finland, pp. 359-364.

- [Tran, 2006] **Mickaël Tran (2006)** *Prolexbase. Un dictionnaire relationnel multilingue de noms propres : conception, implantation et gestion en ligne*. Thèse de doctorat d'informatique, Université François-Rabelais de Tours, 20 octobre 2006, 171 p.
- [Uchida, 1987] **Hiroshi Uchida (1987)** *ATLAS: Fujitsu Machine Translation System*. Machine Translation Summit, September 17-19, 1987, Hakone Prince Hotel, Japan, pp. 129-134.
- [Uchida, 2005] **Hiroshi Uchida (2005)** *UNL language Specification Version 3, edition 3.43*. Technical Report, United Nations University, Tokyo, 86 p.
- [Uchida, Zhu et Senta, 2005] **Hiroshi Uchida, Meiyong Zhu et Tarcisio Della Senta (2005)** *Universal Networking Language*. UNDL Foundation. 218 p.
- [Uchino, Ooyama et Furuse, 1999] **Hajime Uchino, Yoshifumi Ooyama et Osamu Furuse (1999)** *ALTFLASH: a Japanese-to-English Machine Translation System for Stock Market Flash Reports*. MT Summit VII, September 13-17, 1999, Singapore, pp. 438-443.
- [Ueffing, Haffari et Sarkar, 2007] **Nicola Ueffing, Gholamreza Haffari et Anoop Sarkar (2007)** *Semi-supervised model adaptation for statistical machine translation*. Machine Translation, Volume 21, Issue 2 (June 2007), pp. 77-94.
- [Vauquois, 1975] **Bernard Vauquois (1975)** *Some problems of optimization in multilingual automatic translation*. Proc. of First National Conference on the Application of Mathematical Models and Computers in Linguistics, Varna, May 1975, 7 p.
- [Vauquois et Boitet, 1985] **Bernard Vauquois et Christian Boitet (1985)** *Automated Translation at Grenoble University*. Computational Linguistics. vol. 11, n° 1, 1985, pp. 28-36.
- [Veale et Way, 1997] **Tony Veale et Andy Way (1997)** *Gaijin: A Bootstrapping, Template-Driven Approach to Example-Based MT*. International Conference, Recent Advances in Natural Language Processing, Tzigov Chark, pp. 239-244.
- [Verdurand, 2005] **Émile Verdurand (2005)** *Édition générique d'objets linguistiques*. Mémoire de M2R, UJF (Grenoble 1), 115 p.
- [Vigna et Berment, 2002] **Claude Der Vigna et Vincent Berment (2002)** *Ambiguïtés irréductibles dans les langages de mots*. Journées Montoises d'Informatique Théorique, Université Paul Valéry, Montpellier, France, 9-11 Septembre 2002, 3 p.
- [Vo, 2004a] **Trung-Hung Vo (2004a)** *Méthodes et outils pour utilisateurs, développeurs et traducteurs de logiciels en contexte multilingue*. Thèse INPG, Grenoble, 22 décembre 2004, 228 p.
- [Vo, 2004b] **Trung-Hung Vo (2004b)** *SANDO, un outil pour analyser des textes hétérogènes*. JADT 2004 : 7ème Journées internationales d'Analyse statistique des Données Textuelles, pp. 1177-1184.
- [Vo, Phan et Boitet, 2005] **Trung-Hung Vo, Huy-Khanh Phan et Christian Boitet (2005)** *Une composition de solutions génériques pour créer le dictionnaire FEV et importer le vietnamien dans PAPILLON*. Proc. Journées scientifiques LTT, Bruxelles, 8-10 sept. 2005, AUF, P. Thoiron, ed., 9 p.
- [Vu et al., 2005] **Quang-Minh Vu, Laurent Besacier, Éric Castelli et Thi-Ngoc-Yen Pham (2005)** *Extraction automatique de Questions dans les corpus de réunions et de dialogues*. MajeSTIC05, Manifestation des jeunes chercheurs francophones dans les domaines des STIC. Novembre 2005, Rennes, France, 4 p.
- [Wahlster, 2000] **Wolfgang (Ed.) Wahlster (2000)** *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer, 2000, ISBN: 978-3-540-67783-3. 667 p.

[Weaver, 1949] **Warren Weaver (1949)** *Translation*. Machine Translation of Languages: Fourteen Essays. The Technology Press of the Massachusetts Institute of Technology/John Wiley (New York)/Clapham & Hall (London), pp. 15-23.

[Wehrli, 1991] **Éric Wehrli (1991)** *Pour une approche interactive au problème de la traduction automatique*. Proc. Colloque “L’environnement traductionnel. La station de travail du traducteur de l’an 2001”, Mons, 25-27 avril 1991, AUPELF & UREF, pp. 59-68.

[Wehrli, 1992] **Éric Wehrli (1992)** *The IPS System*. Proc. COLING-92, Nantes, 23-28 July 1992, vol. 3/4, pp. 870-874.

[Wehrli, 2004] **Éric Wehrli (2004)** *Traduction, traduction de mots, traduction de phrases*. TALN 2004, Fès, 19–21avril 2004, pp. 483-491.

[Wehrli et Ramluckun, 1993] **Éric Wehrli et Mira Ramluckun (1993)** *ITS-2: an interactive personal translation system*. Proc. of the 6th Conference of the European chapter of the Association for Computational Linguistics (EACL-93), Utrecht, The Netherlands, pp. 476-476.

[Yamamoto, 2004] **Kazuhide Yamamoto (2004)** *Interaction between paraphraser and transfer for spoken language translation*. Journal of Natural Language Processing, 11(5), October 2004, pp. 63-86.

[Yokoi, 1995] **Toshio Yokoi (1995)** *The EDR Electronic Dictionary*. Communications of the ACM, CACM, Volume 38, Issue 11 (November 1995), ACM New York, NY, USA, pp. 42-44.

[Zajac, Casper et Sharpies, 1997] **Rémi Zajac, Mark Casper et Nigel Sharpies (1997)** *An Open Distributed Architecture for Reuse and Integration of Heterogeneous NLP Components*. Proceedings of the fifth conference on Applied natural language processing, 31 March-3 April, 1997, Washington Marriott Hotel, Washington, DC, USA., pp. 245-252.

[Zerfaß, 2002] **Angelica Zerfaß (2002)** *Evaluating translation memory systems*. LREC-2002: Third International Conference on Language Resources and Evaluation. Workshop: Language resources for translation work and research, Las Palmas Canary Islands, 27 May 2002, pp. 49-52.

[Zouhair, 2008] **Teban Zouhair (2008)** *Moniteur interactif Web pour des systèmes de TA et TAO fondés sur UNL*. Rapport de TER d’informatique, UJF, LIG, GETALP, 26 p.

Netographie

[Allen, 2006] **Jeff Allen**, *Documents and references on post-editing MT, website*: <http://www.geocities.com/mtpostediting/>. visité en octobre 2008.

[Boost, 2009] **Boost, Boost Graph Library**, <http://www.boost.org/> visité en octobre 2009.

[Cowlshaw, 2009] **Mike Cowlshaw**, *IBM REXX Brief History*, <http://www-01.ibm.com/software/awdtools/rexx/library/rexxhist.html>. visité en avril 2009.

[Déjà Vu, 2008] **Déjà Vu**, <http://www.atril.com/>. visité en octobre 2008.

[Google-Translation, 2009] **Google-Translation**, http://www.google.fr/language_tools?hl=fr. visité en février 2009.

[GraphML, 2009] **GraphML**, <http://graphml.graphdrawing.org/index.html>. visité en octobre 2009.

[Heartsome, 2009] **Heartsome**, <http://www.heartsome.de/en/termextraction.php>. visité en juillet 2009.

- [IToldU, 2009] **IToldU**, *IToldU*, <http://domus.grenet.fr/itoldu/index.html>. visité en octobre 2009.
- [Jibiki, 2009] **Plate-forme Jibiki**, <http://ligforge.imag.fr/projects/jibiki/>. visité en juillet 2009.
- [KANT, 2009] **Projet KANT**, <http://www.lti.cs.cmu.edu/Research/Kant/>. visité en octobre 2008.
- [Language-Weaver, 2009] **Language-Weaver**, <http://www.languageweaver.com>. visité en mai 2009.
- [Lingua&Machina, 2008] **Lingua&Machina**, <http://www.similis.org>. visité en octobre 2008.
- [Lucy-Software, 2009] **Lucy-Software**, *Lucy Software and Services*, <http://www.lucysoftware.com/>. visité en septembre 2009.
- [Marketplace, 2008] **The Translation Memory Marketplace**, <http://www.tmmarketplace.com/>. visité en octobre 2008.
- [Moses, 2009] **Système Moses**, <http://www.statmt.org/moses/>. visité en février 2009.
- [MultiTerme, 2008] **SDL MultiTerme**, <http://www.sdl.com/en/products/terminology-management/multiTerm.asp>. visité en octobre 2008.
- [Neon, 2009] **Neon**, <http://www.webdav.org/neon/>. visité en octobre 2009.
- [NOOJ, 2009] **NOOJ**, <http://www.nooj4nlp.net/>. visité en octobre 2009.
- [OLIF, 2009] **OLIF**, the *Open Lexicon Interchange Format*, <http://www.olif.net/>. visité en septembre 2009.
- [OMNIA, 2009] **OMNIA**, *Le projet OMNIA*, <http://www.ellemme.org/>. visité en octobre 2009.
- [OpenWFE, 2009] **OpenWFE**, *Open Workflow*, <http://www.openwfe.org>. visité en octobre 2009.
- [Orchestre8, 2009] **Orchestre8**, <http://www.alchemyapi.com/api/keyword/>. visité en juillet 2009.
- [Papillon, 2009] **Papillon**, *Projet Papillon*, <http://www.papillon-dictionary.org/>. visité en octobre 2009.
- [REXX, 2009] **REXX**, *The REXX Language Association*, <http://www.rexxla.org/>. visité en août 2009.
- [STAR-Transit, 2008] **STAR-Transit**, *Transit par STAR*, <http://www.star-group.net/star-www/home/all/star-group/eng/star.html>. visité en octobre 2008.
- [SWIIVRE-UNL, 2004] **SWIIVRE-UNL**, <http://www-clips.imag.fr/geta/User/wang-ju.tsai/welcome.html>. visité en août 2009.
- [SYGMART, 2009] **SYGMART**, <http://www.sygtext.fr/>. visité en octobre 2009.
- [Systèmes-Q, 2009] **Systèmes-Q**, *Démonstration des systèmes-Q*, <http://sway.imag.fr/unldeco/SystemsQ.po?localhost=/home/nguyenht/SYS-Q/MONITEUR/>. visité en octobre 2009.
- [TermServer, 2009] **TermServer**, <http://www.termserver.eu/>. visité en juillet 2009.
- [TRADOS, 2008] **TRADOS**, <http://www.trados.com/en/>. visité en octobre 2008.
- [Translated-Lab, 2009] **Translated-Lab**, <http://labs.translated.net/terminology-extraction/>. visité en juillet 2009.
- [TrM-Translation, 2009] **TrM-Translation**, <http://www.trmkft.hu/en/extract/>. visité en juillet 2009.
- [U++C, 2009] **U++C**, *The U++ Consortium*, <http://www.unl.fi.upm.es/consorcio/index.php>. visité en octobre 2009.
- [UNDL, 2008] **UNDL**, <http://www.undl.org/>. visité en novembre 2008.
- [UNL-RU, 2008] **UNL-RU**, *UNL Russian Language Center*, <http://www.unl.ru/>. visité en novembre 2008.

[UNL-SP, 2008] **UNL-SP**, *Spanish Language Center*,

<http://www.unl.fi.upm.es/english/index.htm>. visité en novembre 2008.

[UNLDeco, 2009] **UNLDeco**, <http://sway.imag.fr/unldeco/Deconvertir.po>. visité en octobre 2009.

[Wiktionary, 2009] **Wiktionary**, *Wiktionary website*,

<http://en.wikipedia.org/wiki/Wiktionary>. visité en juin 2009.

[XML-INTL, 2009] **XML-INTL**, <http://www.xml-intl.com>. visité en février 2009.

[Yakushite.Net, 2008] **Yakushite.Net**, <http://www.yakushite.net/>. visité en octobre 2008.

Annexes

```

19 '': UL('U LOCC').
20 'UNE': UL('UN_ART'), CAT(D), GNR(FEM), NBR(SG).
21 '': UL('U LOCC').
22 'PLACE': UL('PLACE_NF'), CAT(N), GNR(FEM), NBR(SG).
23 '': UL('U LOCC').

AM - Analyse Morphologique

page 2
Résultat de l'exécution, texte : CORPUS ESSAI
code langue : FVX
24 'POUR': UL('POUR_PPCJ'), CAT(S).
25 '': UL('U LOCC').
26 'LE': UL('LE_ART'), CAT(D), GNR(MAS), NBR(SG).
27 'LE': UL('LE_PRON'), CAT(R), GNR(MAS), NBR(SG), PERS(3).
28 '': UL('U LOCC').
29 'QUINZE': UL('QUINZE_CARD'), CAT(CRD).
30 '': UL('U LOCC').
31 'OCTOBRE': UL('OCTOBRE_MS'), CAT(N), GNR(MAS), NBR(SG).
32 '': UL('U LOCC').
33 'A!2': UL('A!2_PP'), CAT(S).
34 '': UL('U LOCC').
35 'NEUF': UL('NEUF_CARD'), CAT(CRD).
36 'NEUF': UL('NEUF_ADJ'), CAT(ADJ), GNR(MAS), NBR(SG).
37 '': UL('U LOCC').
38 'HEURES': UL('HEURE_NF'), CAT(N), GNR(FEM), NBR(PL).
39 '': UL('U LOCC').
40 'SUR': UL('SUR_PP'), CAT(S).
41 'SUR': UL('SUR_ADJ'), CAT(ADJ), GNR(MAS), NBR(SG).
42 '': UL('U LOCC').
43 'UN': UL('UN_ART'), CAT(D), GNR(MAS), NBR(SG).
44 '': UL('U LOCC').
45 'VOL': UL('VOL_NM'), CAT(N), GNR(MAS), NBR(SG).
46 '': UL('U LOCC').
47 'AIR': UL('AIR_NM'), CAT(N), GNR(MAS), NBR(SG).
48 '': UL('U LOCC').
49 'FRANCE': UL('FRANCE'), CAT(INC).

Analyse morphologique.
Mots inconnus, texte: CORPUS ESSAI
Code langue: FVX
1 : FRANCE
19 OCCURENCES pour le texte: CORPUS ESSAI .
1 mots inconnus. ( 1 mots inconnus différents ).
Fin de traitement du texte: CORPUS ESSAI .

Traitement d'un nouveau texte avec les me-mes paramètres:
(chaine d'exécution, contrôle d'exécution, temps): OUI/ NON ?

```

Un autre exemple tiré du système russe-français (RUS-FRA) pour la présentation ergonomique d'arbre

Phase source en transcription:

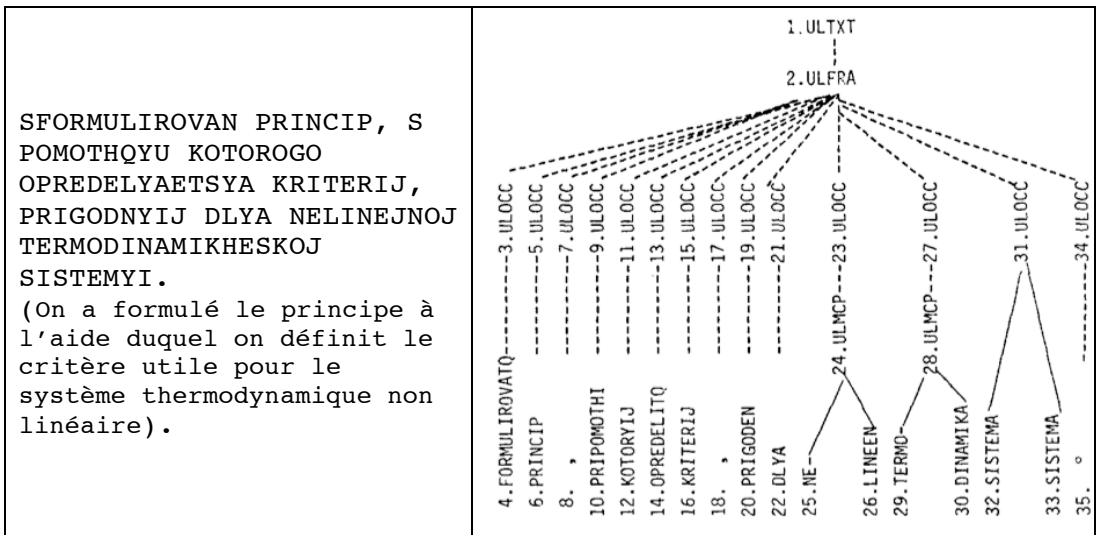


Figure 84: Présentation ergonomique de la sortie d'ATEF

1.2.Exemple de génération morphologique sous SYGMOR

Cet exemple a été extrait du système russe-français (RUS-FRA). L'entrée de cette phase est un arbre produit par la phase GS (génération syntaxique) :

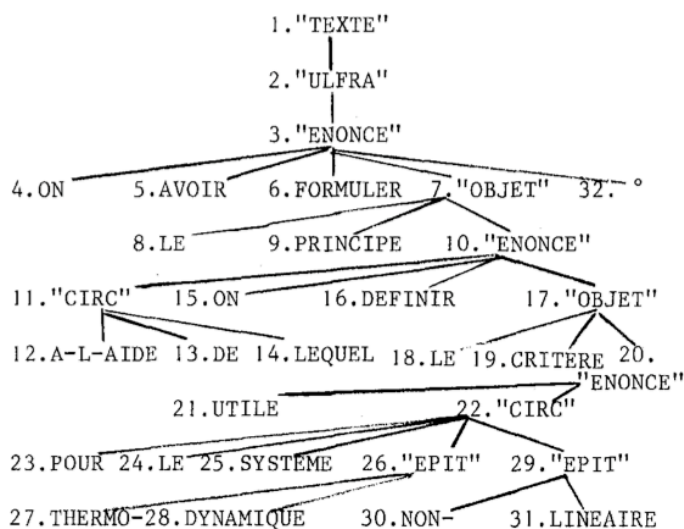


Figure 85: Arbre entrée à SYGMOR

SYGMOR se compose de deux transducteurs. Le premier produit la liste des décorations des feuilles de l'arbre en entrée, et le deuxième transforme cette liste (masques des variables) en une chaîne de caractères de la langue cible, sous le contrôle des données linguistiques. Ces données sont les déclarations de variables, les formats, les procédures, la grammaire et les dictionnaires.

Un article d'un dictionnaire d'UL est de la forme:

nom d'UL ==
[<condition>/affectation/<chaîne>]*/<affectation>/<chaîne>.

Voici des articles pour la génération en français de l'exemple:

A-L-AIDE	==		/ VID / 'A!2 L'AIDE
AVOIR	==	TPIA	/ VID / 'AI,
	==	PSSPT	/ V3H / 'EU,
	==	TP3A	/ V3A / 'A.
LEQUEL	==	NIB	/ VID / 'LAUELLE,
	==	NID	/ VID / 'LESUELLES,
	==	PLU	/ VID / 'LESUELS,
	==		/ VID / 'LEQUEL.

TPIA, PSSPT, TP3A sont les noms de procédure, VID, V3H, V3A sont les noms de format. La sortie de cette phase :

*ON A FORMULE!1 LE PRINCIPE A!2 L'AIDE DUQUEL ON DE!1FINIT LE CRITE!2RE UTILE!1 POUR LE SYSTE!2ME THERMODYNAMIQUE NON LINE!1AIRE.

1.3. Transformation lexicale sous TRANSF

Voici un exemple de dictionnaire bilingue de règle de transfert lexical du système russe-français (RUS-FRA).

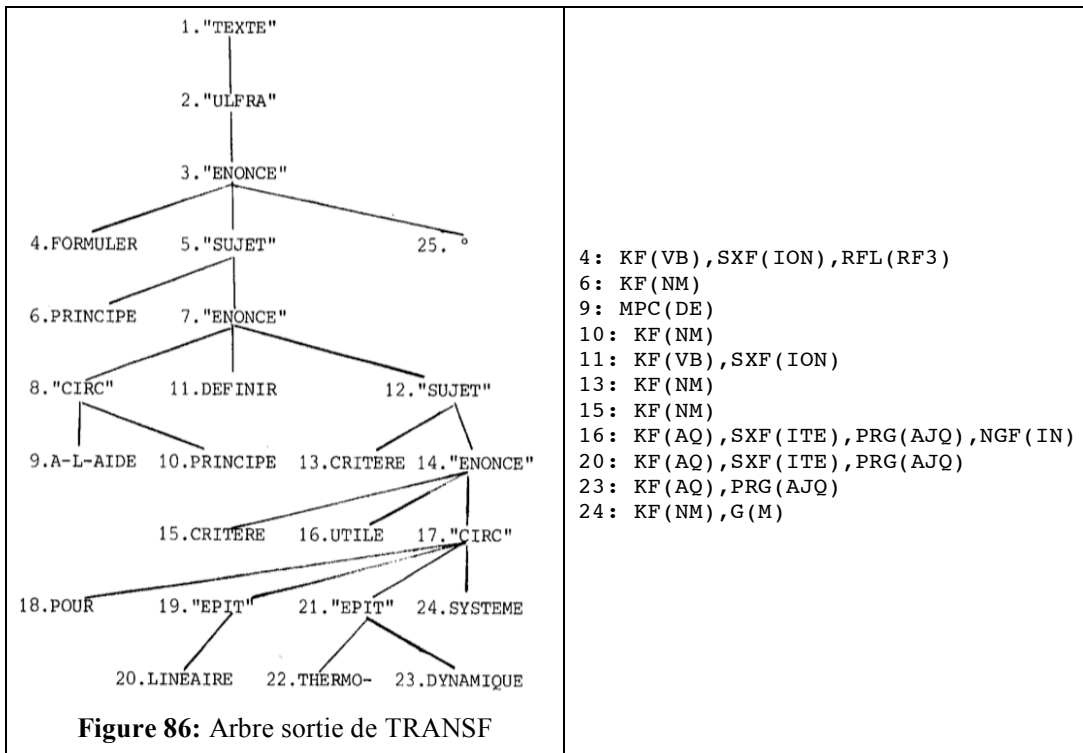
'FORMULIROVATQ'	==	/	/	'FORMULER'	,+VBF1, \$RFPF.
'PRINCIP'	==	/	/	'PRINCIPE'	, \$NMAS.
'PRIPOMOTHI'	==	/	/	'A-L-AIDE'	,+MPCD.
'NAPRIMER'	==	/O(1,2)/		O: 'XLOCF'	,+VIDE ;
				1: 'PAR'	,ZPP ;
				2: 'EXEMPLE'	,XNMMS.

« 0(1,2) » décrit le sous-arbre image créé pour 'NAPRIMER'. Les autres sont réduits à un sommet (défaut).

« +VBF1 » indique que les valeurs de variables nouvelles (par opposition à celles transférées depuis AS) du format VBF1 doivent être recopiées sur le sommet cible. RFPF est une procédure d'affectation, \$note son appel.

« *PP » indique que toutes les variables du format PP (sauf l'UL) sont recopiées sur le sommet 1.

Cette structure est la sortie de cette phase avec l'exemple décrit dans l'annexe 1.1 :



1.4.Exemple d'une réécriture transformationnelle d'un arbre décoré en ROBRA

Cet exemple est tiré de la DSE-1 d'Ariane-G5.

Une règle de transformation se compose d'un nom de règle, d'une partie gauche (le schéma) et d'une partie droite (l'image). Le schéma décrit la famille des sous-arbres que la règle peut transformer. Il se compose d'un modèle géométrique et de conditions sur les variables portées par les sommets. L'image est constituée par le "sous-arbre image", la "fonction de transfert" qui indique où rattacher les dépendants des sommets qui n'apparaissent pas dans le schéma, et la partie "affectation" qui calcule les valeurs des variables portées par les sommets image. Voici un exemple :

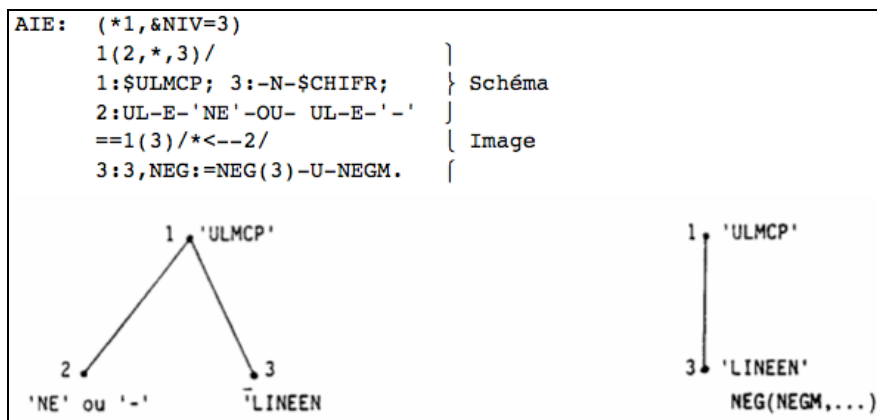


Figure 87: Exemple de transformation d'arbres décorés sous ROBRA

L'exemple ci-dessus représente la règle "AIE", qui traite la négation et s'applique en début d'analyse. Si l'analyse morphologique a produit un mot composé, le sommet correspondant à "NE" est supprimé et l'information de négativité est transférée sur le sommet principal au moyen de la variable NEG.

Le schéma comporte trois sommets, arbitrairement notés 1, 2, 3 (on peut employer tout autre identificateur). Il est représenté à gauche dans l'exemple. L'étoile "*" indique que les sommets 2 et 3 doivent être contigus dans l'arbre objet. Dans l'expression "(*1,&NIV=3)", "&NIV=3" indique que la racine du schéma doit être trouvée dans l'arbre objet au niveau 3 et « *1 » indique que le sommet 1 est invariant dans la transformation de sorte qu'il peut participer à une autre règle. Cette caractéristique est appelée "parallélisme horizontal".

Par exemple, si nous prenons le sous-arbre correspondant à "NELINEJNO-NEPRERYIVNO", nous trouvons deux occurrences de AIE qui peuvent être traitées en parallèle.

Plus généralement, la racine RT de la transformation peut être n'importe quel nœud du schéma. Le sous-arbre de racine RT est la "partie active", le reste étant appelé "contexte". La RT peut elle-même être contextuelle (comme dans cet exemple) ou non. Plusieurs occurrences de règles peuvent partager des sommets contextuels. Les règles qui utilisent cette possibilité sont similaires aux règles "sous-contexte" pour les chaînes.

"1:\$ULMCP" est une condition qui demande que l'UL du sommet 1 soit égale à "ULMCP", en faisant appel à la procédure ULMCP (la confusion est évitée grâce

au préfixe \$), déclarée de la façon suivante, dans la partie `-PROC :`
`"PCP:ULMCP==UL-E'ULMCP"`, `"-E"` étant le relateur d'égalité.

Dans la partie image, `"* ← 2"` signifie que tous les dépendants possibles du sommet doivent être éliminés (transférés sur un sommet "vide"), et `"3:3, NEG := NEG(3)-U-NEGM"` signifie que les variables du nouveau sommet 3 doivent être calculées en prenant les variables de l'ancien sommet 3, puis en ajoutant la valeur NEGM à la variable NEG.

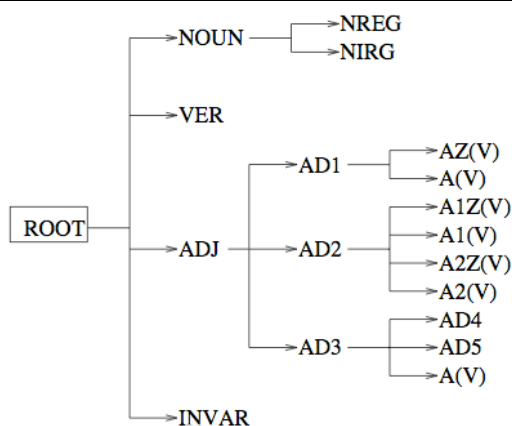
1.5.Exemple d'ATLAS

Cet exemple d'ATLAS [Bachut et Verastegui, 1984] est extrait de la thèse de M. Mangeot [Mangeot, 2001]. Voici le manuel d'indexage source et la forme arborescente pour le manuel papier correspondant. « -- » introduit un commentaire jusqu'à la fin de la ligne (comme // en C++ ou – en ROBRA). ROOT(Q) est un nœud « question » avec 4 arcs adjacents (sortants) correspondant aux réponses possibles. Les commentaires notés entre « ** » et « . » peuvent être écrits sur plusieurs lignes (comme en systèmes-Q et en Ariane-G5).

```

ROOT(Q) : 'type of word to be indexed ?' ;
1 : 'noun'-- NOUN ;
2 : 'verb'-- VERB ;
3 : 'adjective'-- ADJ ;
4 : 'invariant'-- INVAR .
ADJ(Q) : 'what is the adjective type?' ;
** this includes adj with no comp or sup.
1 : 'comp with MORE'-- AD1 ;
2 : 'comp with ER'-- AD2 ;
3 : 'irregular'-- AD3 .
AD1(Q) : 'ambiguous adjective ?' ;
1 : 'yes' -- AZ(V): 'obscure';
2 : 'no' -- A(V): 'expensive'.
AD2(Q) : 'what is the type of the adjective ?' ;
-- type 1 == comp with ER, sup with EST
-- type 2 == comp with ER, sup with ST
-- AMBIGUOUS ie. 'normal ambiguous eg. 'fast'
-- 'normal + comp ambiguous eg. 'light'.
-- 'comp ambiguous eg.
1 : 'type 1, ambiguous'-- A1Z(V): 'light';
2 : 'type 1, non ambig'-- A1(V): 'soft';
3 : 'type 2, ambig'-- A2Z(V): 'saf(e)';
4 : 'type 2, non ambig'-- A2(V): 'unsaf(e)'.
AD3(Q) : 'there are 3 bases to be indexed' ;
1 : 'normal'-- AD4 ;
2 : 'comparative'-- AD5 ;
3 : 'superlative'-- A(V): 'best,driest'.
NOUN(Q) : 'is the noun both regular and variable?' ; -- example of
irregular noun : mouse
1 : 'yes'-- NREG ;
2 : 'no'-- NIRG .

```



Dans ATLAS, on peut faire apparaître les commentaires. Il existe un interpréteur avec une interface ligne ou graphique. On clique sur les nœuds, et à la fin on obtient le code final, qu'on peut envoyer à la bonne place dans l'article courant en cours d'édition (sous XEDIT en Ariane-G5). Un mécanisme semblable a été implémenté par É. Blanc dans CASH, mais sans langage externe et sans possibilité d'imprimer le manuel d'indexage correspondant.

1.6.Exemple de LT

Cet exemple est tiré d'une application [Gaschler et Lafourcade, 1994] de LT [Lepage 1986, Lafourcade 1994] pour la transcription d'entrées du dictionnaire français-anglais-malais à l'USM (Universiti Sains Malaysia). La version montrée ici est la version étendue et « multidialecte » réalisée en MCL (Macintosh Common Lisp) par M. Lafourcade dans le cadre de sa thèse. Elle tourne ici sous Mac OS 9.

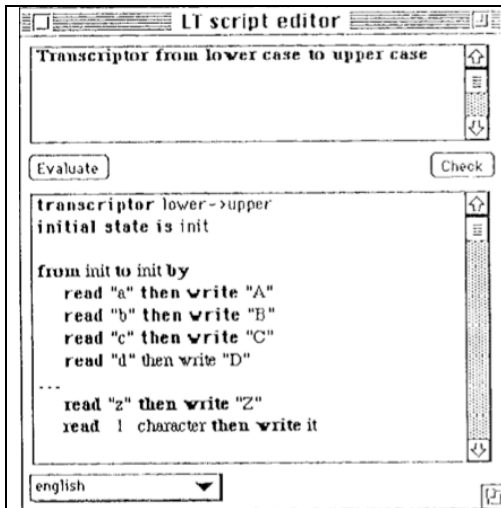


Figure 88: Editeur de LT

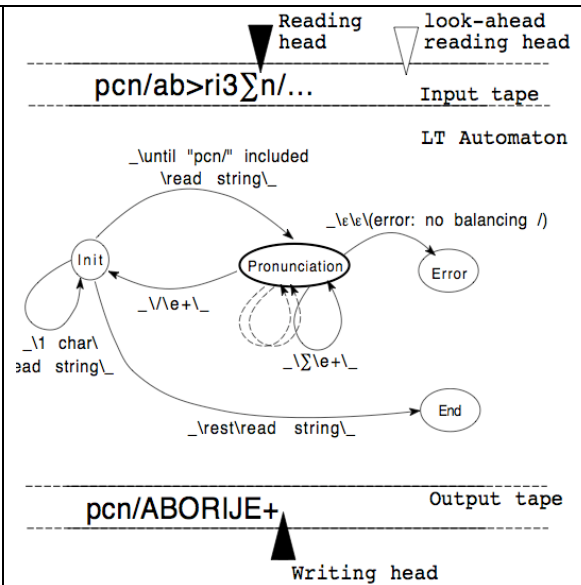


Figure 89: Exemple de LT pour la transcription de formes phonétiques

La conversion est de résoudre le problème d'utilisation des caractères spéciaux dans certain nombre de polices de caractères, en particulier les caractères utilisés à l'USM (police des caractères de Macintosh, ex. Courier ou Times...) dans l'alphabet phonétique international (IPA).

On définit trois formats. Le premier (Ph1) est la forme initiale dans les fichiers MS Word (donné à l'USM). Le deuxième (Ph2) est le format où les caractères spéciaux seront remplacés par les autres, qui sont apparus dans des polices de caractères usuelles. Par l'exemple :

aborigène: /ab>ri3Σn/ → /ABORIE+N/

Pour transcrire de Ph1 à Ph2, on définit le transcripteur Ph1toPh2 suivant :

```

transcriptor Ph1toPh2
initial state is init

from init to init via
  read "Σ" then write "E+"
  read ">" then write "O"
  read "3" then write "J"
  read "e" then write "E-"
  read "â" then write "A-"
  ...
    
```

Le Ph3 est le format phonétique standard d'IPA. Le format standard IPA utilise la police de caractères PalPhon pour représenter les caractères. Par exemple :

aborigène: /ABORIE+N/ → /abɔʀizɛn/

Le transcriptor Ph2toPh3 le passage du format Ph2 au format Ph3.

```
transcriptor Ph2toPh3
initial state is init

from init to init via
  read "E+" then write "{\f1138\`ab}"
  read "O" then write "{\f1138\`bf}"
  read "J" then write "{\f1138\`bd}"
  read "E-" then write "{\f1138 e}"
  read "A-" then write "{\f1138 \`8c}"
...
```

1.7.Exemple de TTEDIT

Cet exemple a été extrait de la thèse de Jean-Claude Durand [Durand, 1988]. Il provient d'un corpus de textes dans lequel l'examen de la typologie avait conduit à exclure *a priori* le traitement de propositions participiales. Or, on a finalement rencontré un cas de ce genre, ce qui a conduit à une construction (structure multiniveau) in correcte.

« les équipements stockés munis de leur cape ».

On a donc édité sous TTEDIT le fichier externe produit par l'analyseur structural. Au moyen d'une règle de réécriture, on a raccroché la structure de la proposition participiale en frère droit du groupe adjectival.

Les affectations sur le nœud « *PPART » ont été faites au moyen de la commande « inputd » sur la décoration.

<pre>PH3S ARBRE A1 V 132 TRUNC=132 SIZE=47 LINE=0 COL=1 ALT=0 FORET ULTXT ULFRA / \ \$\$\$PVB MUNIR-V *GN / \ *GN *GADJ MUNIR-V DE LEUR CAPE-N / \ LE E:1QUIPER STOCKER-V</pre> <p>NOEUD COURANT: ULFRA ====> r: O(1(\$F), 2, 3) == O(1(\$F, PP(2,3))) / PP: ETIQ := "*PPART" .</p>	<pre>PH3S ARBRE A1 V 132 TRUNC=132 SIZE=47 LINE=0 COL=1 ALT=0 ULFRA \$\$\$PVB / \ *GN *GADJ *PPART / \ LE E:1QUIPER STOCKER-V MUNIR-V *GN / \ MUNIR-V DE LEUR CAPE-N</pre> <p>NOEUD COURANT: ULFRA ====> d2fr2zdfrepl "*VK"</p>
<pre>PH3S ARBRE A1 V 132 TRUNC=132 SIZE=47 LINE=0 COL=1 ALT=0 ULFRA \$\$\$PVB / \ *GN *GADJ *PPART / \ LE E:1QUIPER STOCKER-V *VK *GN / \ MUNIR-V DE LEUR CAPE-N</pre> <p>NOEUD COURANT: *VK ====> ufdeco</p>	
<pre>PH3S ARBRE A1 V 132 TRUNC=132 SIZE=47 LINE=0 COL=1 ALT=0 FORET</pre> <p>NOEUD COURANT: FORET ====> input FORME(' '), K(PPART), PERS(3), NB(PLUR), MOD(IND),CIRC(CTXT)</p>	<pre>PH3S ARBRE A1 V 132 TRUNC=132 SIZE=48 LINE=1 COL=1 ALT=2 FORET / FORME K PERS NB MOD CIRC - PPART 3 PLUR IND CTXT</pre> <p>NOEUD COURANT: FORET ====> file</p>

Annexe 2.Exemple de commande envoyée à Ariane-G5 depuis CASH

LIENLANG * * : lister les couples de langues dans l'application.
Commande à envoyer :

<pre><<< >>> - premier enregistrement - ARIANET - LIDIA17 ---. --@@ ABCDEFGHIJKLMNOPQRSTUVWXYZ @@-- ----- zone référence document ----- - ----- fin zone référence document ---- --@@ ABCDEFGHIJKLMNOPQRSTUVWXYZ @@-- ----- zone paramètres ----- MACHINE = CARPENA DISQUE = 191 LGS = * LGC = * ** TRAIT = LIENLANG (* , *) ** ----- fin zone parametres -----</pre>	<p>mode de développement, via LIDIA17 accessible restreint interne sans mot de passe</p>
<pre><<< 19283 >>> - premier enregistrement - ARIANET - LIDIA20 -. /*-----*/ ** Opération de phase: M C ou G , opération composante: M C G ou S. ** 2/5/08 4:31:15 PM . MACHINE = CARPENA DISQUE = 191 LGS = * LGC = * ** TRAIT = LIENLANG (* , *) ** /*-----*/</pre>	<p>mode de déploiement via LIDIA20 avec mot de passe</p>

Résultat reçu:

<pre>/*-----*/ ** Opération de phase: M C ou G, opération composante: M C G ou S. MACHINE = CARPENA DISQUE = 191 LGS = * LGC = * ** TRAIT = LIENLANG (* , *) ** /*-----*/ -> Tout est O.K. /*-----*/ Inexistant XXXX XXX XXX -> Fichier retour inexistant. /*-----*/ -- Pas de spool console modifications. /*-----*/ --- Pas de spool console compilations. --- /*-----*/ Trace de console ARIANE-G5 (traitements). +-----+ ! ! ! ARIANE-G5.3 ! ! ----- ! ! DEBUT DE SESSION ! ! ! +-----+</pre>	<p>Entête de retour avec copie de la commande envoyée et messages d'Ariane. En suite, on trouve la trace de la session Ariane-G5 qui s'est déroulée en mode non interactif (on simule les commandes de l'utilisateur en utilisant la « pile-file » de CMS. La partie utile de ce flot est ci-contre.</p>
<pre>ARIANEG5: type de traitement? -Liens entre les langues- Langue(s) source(s). langue(s) cible(s) associée(s) à cette (ces) langue(s). BAC FAC FAX BAX FAX BA5 FAC BXA FXA</pre>	<p>Information sur les couples de langues à extraire.</p>

<pre> FVX ENX Langue(s) cible(s). langue(s) source(s) associée(s) à cette (ces) langue(s) ENX FVX FAC BAC BA5 FAX BAC BAX FXA BXA </pre>	
<pre> ARIANEG5: type de traitement? +-----+ ! ! ! ARIANE-G5.3 ! ! ----- ! ! FIN DE SESSION ! ! ! +-----+ Voulez-vous votre mouchard de session: OUI/NON/FERMER/CONT/?/DET ? /*-----*/ </pre>	Fin de la session Ariane-G5.

Annexe 3. Grammaire syntaxique en ANTLR pour compiler les grammaires-Q et les graphes-Q

```

/*****
**
* Grammaire en ANTLR des systèmes-Q pour ANTLR V3.
* CB et JCD, Bangkok, 01/11/2007.
* CB & NHT, Grenoble, 22/11/2007: un axiome commun pour grammaires et graphes
* NHT, Grenoble, 14/12/2007: ajout des actions de chargement dans la structure interne
* NHT, Grenoble, 10/01/2009: Commentaires
*****/
grammar SysQ;
// mettre grammar SysQi pour ANTLRworks
options
{
    language=C;
}
@header
{
    #include "SysQ_chaine.h"
    #include "SysQ_arbre.h"
    #include "SysQ_graphe.h"
    #include "SysQ_gram.h"
    #include "SysQ_global.h"
    #include "../MONITEUR/SysQ_message.h"
}
@lexer::header {
    #define SYSQ_CHANNEL_COMMENT 1 // 0: DEFAULT, 99: HIDDEN, le canal DEFAULT
    qui sera reconnu par parser (l'analyseur), les autres NON.
}

sysqGraphGram : // CB 27/10/07 pour que tout soit accessible depuis 1 axiome
    fichDonneesQ // mais on continue a ne pas mélanger des graphes et
    ! fichTraitementsQ // des règles dans un même fichier.
    ;

fichDonneesQ :
    donneesQ
    ;

fichTraitementsQ:
    traitementsQ
    ;

BLANCS
@option{ warning=false; } : // Pour ANTLR ne dit plus sur la redondance
(
    ! ('\\r\\n') // MS
    ! '\\n' // Unix
    ! '\\r' // Mac
)+
{ $channel=HIDDEN; }
;

Commentaire:

```

```

STAR STAR (~ ( '.' ))* '.'
    { $channel=SYSQ_CHANNEL_COMMENT; }
;

REQ:      '-REQ-'      ;
DET:      '-DET-'      ;
INV:      '-INV-'      ;
OU:       '-OU-'       ;
ET:       '-ET-'       ;
NON:      '-NON-'      ;
DANS:     '-DANS-'     ;
HORS:     '-HORS-'     ;
NUL:      '-NUL-'      ;
NE:       '-NE-'       ;

// signes significatifs: (le vrai signe "différent" n'est pas iso8859-1)
// *****
PLUS:     '+'          ;
TIRET:    '-'          ;
STAR:     '*'          ;
EXP:      '^'          ;
SLASH:    '/'          ;
PARG:     '('          ;
PARD:     ')'          ;
DOLLAR:   '$'          ;
EG:       '='          ;
DIFF:     '#'          ;
POINT:    '.'          ;
VIRG:     ','          ;

// signes non significatifs:
// *****
// Il manque les signes: "congruence", "ou", "et", et "non" logiques, les fleches
// "haut", "bas", "angle droit a droite", "<=" et ">=".
CRG:      '['          ;
CRD:      ']'          ;
DEUXPTS:  ':'          ;
INF:      '<'          ;
SUP:      '>'          ;
PVIRG:    ';'          ;
BARRE:    '!'          ;

CHIFFRE   :   '0'..'9';

ABCDEF    :   'A'..'F';
IJKLMN   :   'I'..'N';
UVWXYZ   :   'U'..'Z';
G        :   'G';
H        :   'H';
OPQRST   :   'O'..'T';

LETTRE_MIN :   'a' .. 'z';

UNICODE_CHAR: '\\\ 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT;

fragment // Token utilisé en interne dans d'autres règles lexicales ou syntaxiques
HEX_DIGIT : '0'..'9'!'a'..'f'!'A'..'F';

lettre returns [char ch]:
    let=(ABCDEF
        ! G
        ! H
        ! IJKLMN
        ! OPQRST
        ! UVWXYZ
        ! LETTRE_MIN) { $ch=$let.text->chars[0];}
;

signeSignificatif returns [char ch]:
    sSign=( PLUS
        ! STAR
        ! TIRET
        ! SLASH
        ! PARG
        ! PARD
        ! DOLLAR
        ! DIFF
        ! VIRG

```

Annexes

```
! EXP
! POINT ) {$ch = $$sSign.text->chars[0];}
;

// suppression du caractere 'non' sinon ne passe pas !
signeNonSignificatif returns [char ch]:
  sNSign=( CRG
! CRD
! DEUXPTS
! INF
! SUP
! PVIRG
! BARRE )
  {$ch = $$sNSign.text->chars[0];}
;

caractere returns [char ch]:
  let=lettre           {$ch = $let.ch; }
!   chif=CHIFFRE      {$ch = $chif.text->chars[0]; }
!   sSign=signeSignificatif {$ch = $$sSign.ch; }
!   sNSign=signeNonSignificatif {$ch = $$sNSign.ch; }
;

caractereNonPointOuEgal returns [char ch]:
  carac=( PLUS
! STAR
! TIRET
! SLASH
! PARG
! PARD
! DOLLAR
! DIFF
! VIRG
! EXP )
! l=lettre           {$ch = $carac.text->chars[0];}
! chif=CHIFFRE      {$ch = $l.ch;}
! sSign=signeSignificatif {$ch = $chif.text->chars[0];}
! sNSign=signeNonSignificatif {$ch = $$sNSign.ch;}
;

// Fin de la partie lexicale, realisee en partie au niveau syntaxique
// -----

// Donnees-Q = liste de graphes de chaines d'arbres simplement etiquetes
// -----
donneesQ returns [int idxDonneeQ]:
  gdc=grapheDeChaines {
    T_Graphe graphe;
    graphe.entree = $gdc.idxGrappeDeChaines;
    graphe.sortie = getLastNoeudChemin($gdc.idxGrappeDeChaines).numero;
    $idxDonneeQ = addGrappe(graphe);
  }
  ( donneesQ )?
;

grapheDeChaines returns [int idxGrappeDeChaines]:
  cS=cheminSimple { $idxGrappeDeChaines = $cS.cSimple; noLine=LT(1)->line; }
  ( SLASH gdc=grapheDeChaines)?
;

cheminSimple returns [int cSimple]:
  (sDebut=sommet ch=chaîne sFin=sommet)
  { $cSimple = addCheminSimple($sDebut.val, $ch.idxChaine, $sFin.val); }
;

sommet returns [int val]:
  tiretDeb=TIRET ent=entier {$val = $ent.val; } tiretFin=TIRET
;

entier returns [int val]
  @init{ char s[] = "AAAA";}:
  c1=CHIFFRE {s[0] = $c1.text->chars[0];}
  (c2=CHIFFRE {s[1]=$c2.text->chars[0];}(c3=CHIFFRE{s[2]=$c3.text->chars[0];})? )?
  { sscanf(s, "%dA", &$val); }
;

chaîne returns [int idxChaine] // NoeudGrappe et ArcGrappe ici
  @init { int iArc = -1; }:
  ar=arbre { T_NoeudGrappe noeudGrappe; noeudGrappe=noeudNull; $idxChaine =
```

```

addNoeudGraphe(noeudGraphe); T_ArcGraphe arc; arc.noeudArbre=$ar.idxArbre;
arc.noeud=$idxChaine; iArc=addArcGraphe(arc); }
  ( PLUS ch=chaine { updateArcNoeuds($idxChaine, iArc, $ch.idxChaine); } )?
;

arbre returns [int idxArbre] // Retourner idx dans tbNoeudArbres vient d'etre ajoute
@init { T_NoeudArbre noeudArbre; int hasFilsG = 0; }
@after { }:
  etiq=etiquette { noeudArbre.R_Chaine = $etiq.R_Chaine; noeudArbre.numPar=FALSE;
noeudArbre.benjamin=TRUE; }
  ( PARG l=liste { hasFilsG = 1;} PARD )?
  { noeudArbre.filsG=hasFilsG==1?$l.debut:_NIL; $idxArbre =
addNoeudArbre(noeudArbre); }
;

liste returns [int debut] // retourner le debut de la liste
@init { int isLast = 1; T_NoeudArbre *pNoeudArbre; }
@after { }:
  ar=arbre { pNoeudArbre = &tbNoeudsArbre[$ar.idxArbre]; $debut = $ar.idxArbre; }
  ( VIRG l=liste {isLast = 0; pNoeudArbre->frereD = $l.debut;} )?
  { pNoeudArbre->benjamin=isLast==1?TRUE:FALSE; }
;

/*listeSimple: arbre ;*/

commentaire:
  (caractereNonPoint)* POINT
  texte POINT //TODO: garder la position des commentaires
  //et commentaires dédiés pour un objet
;

texte:
  STAR STAR
  ( caractereNonPointOuEgal )*
;

// Traitements-Q = liste de grammaires-Q (ou "phases")
// -----

traitementsQ returns [int idx]:
  ph=phase { $idx=$ph.idx; noLine=LT(1)->line; }
  ( REQ traitQ=traitementsQ { int iR=setTraitementQSuivant($idx, $traitQ.idx); }
)?
;

phase returns [int idx]
@init{ int det=0; int inv=0; } :
  ((DET {det=1;})? (INV{inv=1;})? sysQ=systemeQ) {
$idx=addTraitementQWithIdxNoLine(det, inv, $sysQ.idx, noLine); }
;

systemeQ returns [int idx]
@init{ $idx = _NIL; }:
  ( r=regle { if ($idx==_NIL) $idx=$r.idx; } ) *
;

regle returns [int idx]
@init{ }:
  (mG=membreGauche EG EG mD=membreDroit cd=condition POINT){
noLine=LT(1)->line; $idx=addRegleWithIdNoLine($mG.idx,$mD.idx,$cd.idx,noLine); }
;

membreGauche returns [int idx]:
  chParam=chaineParametree { $idx=$chParam.idx; }
  ! (TIRET TIRET) { $idx=addNoeudTiretTiret(); }
;

membreDroit returns [int idx]:
  chParam=chaineParametree { $idx=$chParam.idx; }
  ! (TIRET TIRET) { $idx=addNoeudTiretTiret(); }
;

chaineParametree returns[int idx]
@init{int isLast;} :
  ( arParam=arbreParametre { $idx=$arParam.idx; isLast=1; }
  ( p=PLUS chParam=chaineParametree { int iR=updateFrereDPlus($idx, $chParam.idx);
isLast=0; } )?)
  { if (isLast==1) {int iR=updateFrereDPlusLast($idx); }}

```


Annexes

```
;

arbreParametre returns [int idx]
@init { int contientFilsG=0; }:
( paramArbre=parametreArbre { $idx=addParametreArbre($paramArbre.R_Chaine); }
!   etiqParam=etiquetteParametree {
$idx=addEtiquetteParametree($etiqParam.R_Chaine, $etiqParam.type); }
( PARG listeParam=listeParametree { contientFilsG=1; } PARD )?
) { if (contientFilsG==1) {int iR=updateFilsG($idx, $listeParam.idx); } }
;

listeParametree returns[int idx]
@init{ int isBenjamin=1; }:
( listeSParam=listeSimpleParametree { $idx=$listeSParam.idxNoeudArbre; }
( VIRG listeParam=listeParametree { isBenjamin=0; } )?
) { if (isBenjamin==1) {int iR=updateBenjamin($idx);} else {int
iR=updateFrereD($idx, $listeParam.idx);} }
;

listeSimpleParametree returns [int idxNoeudArbre]
@init { }:
paramList=parametreListe {
$idxNoeudArbre=addParametreListe($paramList.R_Chaine); }
!   arbreParam=arbreParametre { $idxNoeudArbre=$arbreParam.idx; }
!   DOLLAR DOLLAR etiqParam=etiquetteParametree {
$idxNoeudArbre=addG_etiquetteParametree($etiqParam.R_Chaine, $etiqParam.type); }
!   NUL { /*Rien*/ }
;

etiquetteParametree returns [int R_Chaine, int type]:
paramEtiquette=parametreEtiquette {
    $R_Chaine = $paramEtiquette.R_Chaine; $type=_A;}
!   etiq=etiquette { $R_Chaine = $etiq.R_Chaine; $type=_E; }
;

condition returns[int idx]
@init{ $idx=_NIL;}:
( SLASH expBool=expressionBooleenne { $idx=$expBool.idx; } )?
;

expressionBooleenne returns [int idx]
@init { int hasOU = 0;}:
secondBool=secondaireBooleen
( OU expBool=expressionBooleenne {hasOU=1;} )?
{ if (hasOU==0) $idx=$secondBool.idx; else
$idx=addCond_OU_expressionBooleenne($secondBool.idx, $expBool.idx); }
;

secondaireBooleen returns [int idx]
@init{ int hasET=0; }:
primBool=primaireBooleen
( ET secondBool=secondaireBooleen { hasET=1;} )?
{ if (hasET==0) $idx=$primBool.idx; else
$idx=addCond_ET_secondaireBooleen($primBool.idx, $secondBool.idx); }
;

primaireBooleen returns [int idx]:
( TIRET TIRET ) { $idx=addConditionTiretTiret(); }
!   (NON primBool=primaireBooleen) {
    $idx=addCond_NON_primaireBooleen($primBool.idx); }
!   (PARG POINT expBool=expressionBooleenne POINT PARD) {
    $idx=addCond_exp_primaireBooleen($expBool.idx); }
!   (lParamG=listeParametree rela=relation lParamD=listeParametree) {
    $idx=addCondRelationListeParams($lParamG.idx, $rela.rel, $lParamD.idx);}
;

relation returns[int rel]:
EG { $rel=EG; }
!   DIFF { $rel=DIFF; }
!   DANS { $rel=DANS; }
!   HORS { $rel=HORS; }
!   NE { $rel=NE; }
;

indice returns [char ch]
@init{ ch = '\0'; }:
( chiff=CHIFFRE {ch = $chiff.text->chars[0];} )?
;
```

```

etiquette returns[int R_Chaine]
  @init{ char *eti = getNString(LENG_OUT_TEXT); }
  @after{ }:
  carac=caractere (
let=lettre {addLettre(stiq, $carac) ;}
! chiff=CHIFFRE { addChiffre(eti,$chiff) ; })*
  { $R_Chaine = addChaine(eti); free(eti); eti=NULL; }
;

parametreEtiquette returns [int R_Chaine]:
  (abcdef=ABCDEF s=STAR ind=indice)
  { $R_Chaine = addParametreObjet($abcdef.text->chars[0], $s.text->chars[0],
$ind.ch); }
;

parametreArbre returns [int R_Chaine]:
  (ijklmn=IJKLMNOP s=STAR ind=indice)
  { $R_Chaine = addParametreObjet($ijklmn.text->chars[0], $s.text->chars[0],
$ind.ch); }
;

parametreListe returns [int R_Chaine]:
  (uvwxyz=UVWXYZ s=STAR ind=indice)
  { $R_Chaine = addParametreObjet($uvwxyz.text->chars[0], $s.text->chars[0],
$ind.ch); }
;

```

Annexe 4.Trace d'exécution d'une maquette de TA vietnamien-français par les systèmes-Q

Il s'agit une maquette de TA pidgin vietnamien-français pour une phase simple avec une ambiguïté.

Phase source originale : Ông già đi nhanh quá.

Phase transcrite sans les accents : ONG + GIA + DI + NHANH + QUA

Cela peut être compris comme :

(ONG GIA) + DI + (NHANH QUA) == LE VIEUX VA TROP VITE

ou bien :

(ONG) + (GIA DI) + (NHANH QUA) == TU VIEILLIS TROP VITE

Données-Q : SYSQ_DATA_VNFR.TEST
-01- ONG + GIA + DI + NHANH + QUA -02-
Traitements-Q : SYSQ_GRAM_VNFR.TEST
<pre> ** REGROUPEMENT EN ARBRES . ONG == MOT(ONG). ONG + GIA == MOT(ONG,GIA). GIA == MOT(GIA). GIA + DI == MOT(GIA, DI). DI == MOT(DI). NHANH + QUA == MOT(NHANH, QUA). ** TRANSFORMATION EN AJOUTANT DES FONCTIONS LEXICALES. ** PREMIERE POSSIBILITE. ** DICTIONNAIRE D'ANALYSE. MOT(ONG, GIA) == AMV(LM(ONG, GIA), CAT(S), SUBR(PRON), PERS(2), NBR(SING), FR(LM(VIEUX))). MOT(DI) == AMV(LM(DI), AT(R), CAT(V), PERS(2), NBR(SING), FR(LM(VA))). MOT(NHANH, QUA) == AMV(LMC(NHANH, QUA), (CAT(A), SUBA(ADV), FR(GADV(VITE,(TRES,*))). ** DEUXIEME POSSIBILITE. MOT(ONG) == AMV(LM(ONG), CAT(R), SUBR(PRON), PERS(2), NBR(SING), FR(LM(TU))). MOT(GIA, DI) == AMV(LM(GIA, DI), AT(R), CAT(V), PERS(2), NBR(SING), FR(LM(VIEULLES))). MOT(NHANH, QUA) == AMV(LMC(NHANH, QUA), (CAT(A), SUBA(ADV), </pre>

```

FR(GADV(VITE,(TRES,*))).
** GENERATION EN FRANÇAIS.
-REQ-
  AMV(I*1, U*1, FR(V*1)
+  AMV(I*2, U*2, FR(V*2)
+  AMV(I*3, U*3, FR(V*3))
== PHVB(GSUJ(LMFR(V*1), VIET(U*1)),
        NV(LMFR(V*2), VIET(U*2)),
        GMAN(LMFR(V*3, VIET(U*3)))
/  SUBR(PRON) -DANS- U*1
-ET- CAT(V)   -DANS- U*2
-ET- SUBA(ADV) -DANS- U*3.

```

Voici le résultat résumé sur chaque phase:

- Après l'analyse :

```

-+1/_1- MOT(ONG) -+2/N2-
-+2/N2- MOT(GIA) -+3/N3-
-+3/N3- MOT(DI) -+4/N4-
-+4/N4- MOT(NHANH,QUA) -+6/_2-
-+1/_1-MOT(ONG,GIA) -+3/N3-
-+2/N2-MOT(GIA,DI) -+4/N4-
-+4/N4-MOT(NHANH) -+5/N5-
-+5/N5-MOT(QUA) -+6/_2-

```

- La première possibilité :

```

-+1/_1- AMV(LM(ONG, GIA), CAT(S), SCAT(PRON), PERS(2), NBR(SING),
FR(LM(VIEUX)) -+3/N3-
-+3/N3- AMV(LM(DI), AT(R), CAT(V), PERS(2), NBR(SING), FR(LM(VA)) -
+4/N4-
-+4/N4- AMV(LMC(NHANH, QUA), CAT(A), SUBA(ADV),
FR(GADV(VITE,(TRES,*))) -+6/_2-

```

- La deuxième possibilité :

```

-+1/_1- AMV(LM(ONG), CAT(R), CAT(PRON), PERS(2), NBR(SING),
FR(LM(TU)) -+2/N2-
-+2/N2- AMV(LM(GIA,DI), AT(R), CAT(V), PERS(2), NBR(SING),
FR(LM(VIEULLES)) -+4/N4-
-+4/N4- AMV(LMC(NHANH, QUA), (CAT(A), SUBA(ADV),
FR(GADV(VITE,(TRES,*))) -+6/_2-

```

- Après la génération :

```

-+1/_1- PHVB(
  GSUJ(LMFR(LM(VIEUX)), VIET(CAT(S), SCAT(PRON), PERS(2), NBR(SING))),
  NV(LMFR(LM(VA)), VIET(AT(R), CAT(V), PERS(2), NBR(SING))),
  GMAN(LMFR(GADV(VITE,(TRES,*)), VIET(CAT(A), SUBA(ADV)))
-+6/_2-

```

```

-+1/_1- PHVB(
  GSUJ(LMFR(LM(TU)), VIET(CAT(R), CAT(PRON), PERS(2), NBR(SING))),
  NV(LMFR(LM(VIEULLES)), VIET(AT(R), CAT(V), PERS(2), NBR(SING))),
  GMAN(LMFR(GADV(VITE,(TRES,*)), VIET(CAT(A), SUBA(ADV)))
-+6/_2-

```