Transfert d'applications linguistiques du système ARIANE au système SYGMART

Francis Brunet-Manquat & Olivier Guyotot
Tuteur : Gilles Sérasset
GETA[1] - CLIPS[2]

23 septembre 1999

[1]Groupe d'Étude pour la Traduction Automatique
[2]Communication Langagière et Interaction Personne Système

Table des matières

Table des figures

Remerciements

Nous remercions toute l'équipe du GETA pour nous avoir accueillis pendant cette année de magistère et pour son soutient permanent.

Nous tenons à adresser plus particulièrement nos remerciements à :

- **Gilles Sérasset**, pour nous avoir accepté comme stagières en magistère et pour son aide tout au long de l'année
- **Jean-Phillipe Guilbauld**, pour son aide précieuse sur ARIANE
- **Étienne Blanc**, pour ses explications sur l'application UNL
- **Karen Fort**, pour son aide à la compréhension du langage UNL

ainsi qu'à tous nos amis de maîtrise.

Introduction

Voici bientôt 36 ans que les recherches en traduction automatique et assistée par ordinateur (TA, TAO) se poursuivent à Grenoble. L'équipe GETA (Groupe d'Étude pour la Traduction Automatique) du laboratoire CLIPS (Communication Langagière et Interaction Personne Système) participe activement à ces recherches. C'est une équipe pluridisciplinaire formée d'informaticiens et de linguistes. Les thèmes de recherche du GETA concernent tous les aspects théoriques, méthodologiques et pratiques de la TAO, et plus généralement de l'informatique multilingue. Une des plus grandes réalisations du GETA fut le premier "générateur de systèmes de TAO" complet, le système ARIANE. Il fut conçu entre 1975 et 1979. Son noyau principal a été programmé en assembleur. Ce système fonctionne sur un IBM 360/67 (la première machine à mémoire virtuelle). Après une deuxième version sortie en 1985, c'est encore aujourd'hui sous le système ARIANE que sont développées les applications linguistiques du GETA. Un des concepteurs du système ARIANE, J. Chauché, réalisa en 1990, un système fonctionnant sur les mêmes principes qu'ARIANE, le système SYGMART, écrit en C et fonctionnant sur des petits systèmes UNIX.

L'ancienneté du système ARIANE commence à devenir un handicap important à son utilisation : coûts de fonctionnement de l'IBM prohibitifs, lenteur d'exécution, difficultés de la maintenance et d'améliorations... tout contribue à envisager un terme à son fonctionnement. Mais il existe encore aujourd'hui peu de systèmes aussi "performants" pour la traduction automatique. Envisager la mise au point d'une nouvelle version aujourd'hui est bien sûr possible, mais c'est un projet de longue haleine et les bénéfices d'une telle entreprise sont difficilement mesurables. D'où l'idée de tenter de porter les applications existantes écrites pour ARIANE vers un autre système. En raison d'une origine presque commune et des similitudes importantes dans le fonctionnement, SYGMART paraît être le candidat parfait. En dépit de l'absence d'une interface vraiment conviviale, c'est un système résolument plus récent qu'ARIANE, avec des limitations de mémoire moins strictes et surtout l'avantage d'être "portable", puisqu'il peut fonctionner sur un système UNIX, quelque soit l'architecture matérielle sous-jacente.

Une des applications existante sous ARIANE aujourd'hui est l'application UNL (Universal Networking Language). C'est l'application que nous avons choisie de porter sous SYGMART pour valider nos résultats. En outre, Hewlett Packard s'intéresse au langage UNL, et disposer d'une application fonctionnelle

sous SYGMART permettrait de faciliter le développement d'une maquette destinée à tester l'UNL sur les textes HP.

Dans la première partie de ce rapport, nous présentons notre étude du transfert d'applications du système ARIANE au système SYGMART. Plus précisément, la problématique était de savoir s'il est possible de transferer *automatiquement* toutes les applications d'un système vers l'autre. Dans la seconde partie, nous nous sommes intéressés plus particulièrement à l'application UNL à la fois pour valider les résultats de notre étude et pour adapter cette application aux textes de HP.

Première partie

Étude du transfert automatique d'applications du système ARIANE au système SYGMART

Chapitre 1

Contexte

Les systèmes ARIANE et SYGMART ont été conçus pour la traduction automatique. Ces systèmes permettent à des linguistes de developper des applications linguistiques (linguiciels), grâce à des langages spécialisés de manipulations de structures arborescentes. Ces manipulations ont pour but essentiel la reconnaissance de relations entre les diverses parties d'un texte. Nous allons tout d'abord vous présenter l'organisation des systèmes ARIANE et SYGMART, puis nous décrirons les éléments manipulés par le système SYGMART. Enfin, nous aborderons les différentes fonctions (ou langages) composant le système SYGMART.

1.1 "Nos" systèmes de traduction automatique

La traduction automatique (TA) est organisée en trois phases logiques successives, une analyse monolingue, un transfert bilingue et une génération monolingue (figure 1.1). La première phase consiste à analyser le texte source et à produire un élément (une structure linguistique) correspondant au texte et à la langue associée. Cette structure peut avoir différentes représentations. La seconde phase consiste à manipuler cet élément structuré pour le transformer en un autre élément correspondant à la langue cible. Enfin, grâce à cette structure, le système peut effectuer la troisième et dernière phase, la génération du texte cible. Pour réaliser ces trois phases, nos systèmes comportent trois grandes fonctions chacun (figure 1.2), nous pouvons également parler de sous-systèmes ou de langages :

- Un analyseur morphologique (basé sur un modèle d'automates à états fini) : cette fonction permet de passer d'une chaîne de caractères à un élément structuré. (OPALE pour SYGMART / ATEF pour ARIANE)
- Une fonction de transformation d'arbres : elle permet de manipuler les éléments structurés. (TELESI / ROBRA)
- Une fonction de composition de morphèmes : elle permet de définir une chaîne de caractères à partir d'un élément structuré. (AGATE / SYG-

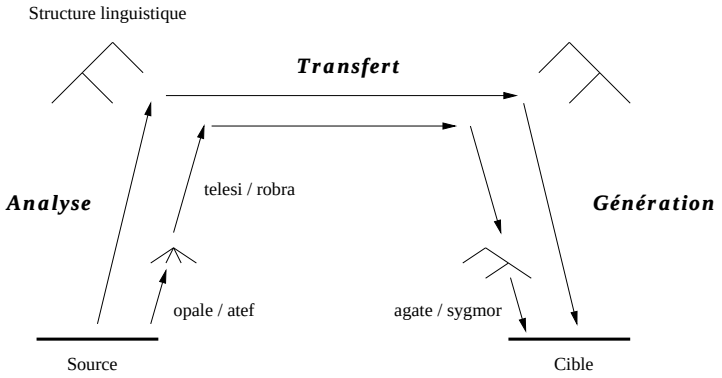


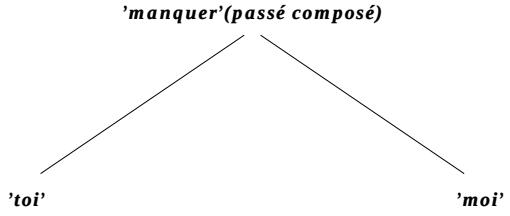
FIG. 1.1: Les différentes phases d'une traduction

MOR)

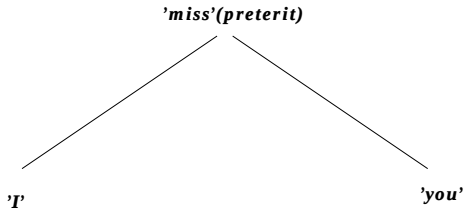
Il faut bien comprendre que ces langages ne correspondent pas aux phases logiques de la traduction. La phase d'analyse utilise les deux premières fonctions et la phase de génération les deux dernières.

Exemple de traduction français - anglais : "**tu m'as manqué**"

- Analyse du texte et production d'une structure linguistique possible du français :



- Transformation de la structure obtenue après analyse en une structure linguistique anglaise (noter l'inversion des arguments) :



- Texte anglais obtenue après génération : "**I missed you**"

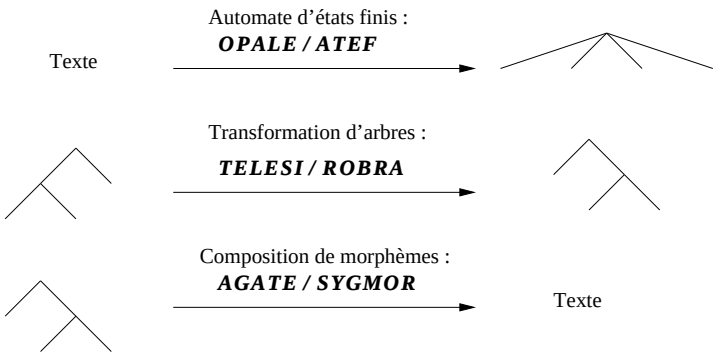


FIG. 1.2: Organisation des langages des systèmes ARIANE et SYGMART

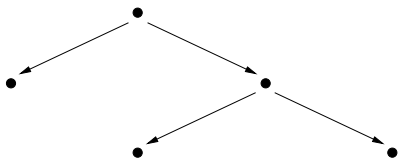
1.2 Les éléments manipulés par le système SYGMART

L'approche d'un élément structuré s'effectue en trois étapes. Tout d'abord nous allons décrire l'arborescence utilisée, puis l'arborescence étiquetée et nous verrons enfin la structure des éléments manipulés par le système SYGMART.

Description d'une arborescence

Une arborescence est un graphe ayant des propriétés particulières :

- Chaque noeud a un et un seul antécédent
- La racine n'a aucun antécédent

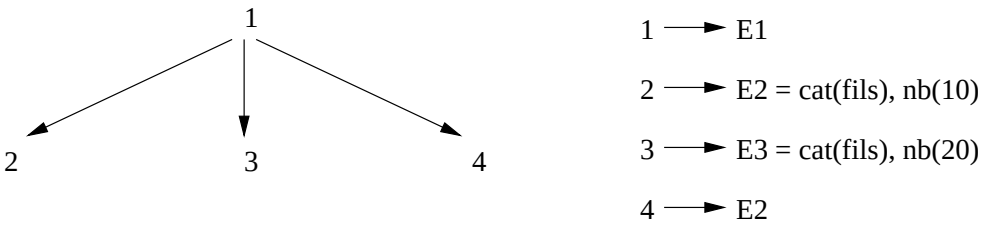


Description d'une arborescence étiquetée

Plusieurs approches sont possibles pour définir une arborescence étiquetée. Toutes les définitions ont pour but d'associer aux points d'une arborescence un renseignement particulier, une étiquette. L'ensemble des renseignements possibles constitue l'univers de référence des étiquettes. Une étiquette est donc un masque de variable ou, plus exactement, un objet pouvant contenir un ensemble

de variables affectées de valeurs.

Exemple d'arborescence étiquetée :



Une arborescence étiquetée est donc un triplet (A, E, f) où A est une arborescence, E un ensemble d'étiquettes, et f une fonction de A dans E.

Élément structuré

Un élément structuré est un triplet (E, S, f) où :

- E est un ensemble fini d'étiquettes
- S est un ensemble d'arborescences
- f est une fonction d'étiquetage de $\text{Noeud}_s \rightarrow \text{E}$

Ainsi l'élément structuré est un ensemble d'arborescences étiquetées, chaque arborescence représentant une "dimension". Nous n'aborderons pas l'aspect multi-dimension des éléments structurés du système SYGMART dans ce rapport pour la simple raison que les élément du système ARIANE n'ont qu'une seule dimension.

1.3 Le système SYGMART

Les objets manipulés par le système SYGMART sont plus complexes que des arborescences simples mais la façon de les appréhender et de les manipuler se fera par l'intermédiaire de transformations d'arborescences. Le système SYGMART est un système de programmation, il est composé de trois langages (sous-systèmes) correspondant aux fonctions élémentaires à réaliser. Le langage OPALE réalise le passage entre les chaînes de caractères d'entrée et les éléments structurés, le langage TELESI réalise les manipulations des éléments structurés et le langage AGATE réalise le passage entre un élément structuré et une chaîne de caractères. On peut considérer le système SYGMART comme le système TELESI auquel on rajouterait deux fonctions OPALE et AGATE pour permettre la réalisation d'entrée-sortie.

1.3.1 Le langage OPALE

Le but du langage OPALE est de définir une transition entre un texte d'entrée et un élément structuré. Ce langage peut être considéré comme une fonction

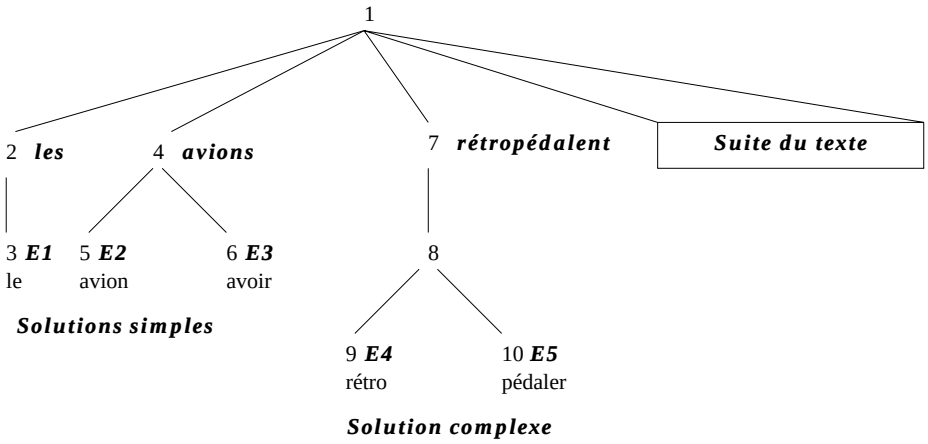


FIG. 1.3: Élément structuré résultant d'une analyse OPALE

d'entrée du système. L'analyse OPALE est effectuée par un transducteur non déterministe d'états finis. Nous construisons l'étiquette d'un élément structuré par la consultation d'un dictionnaire. Ce transducteur fournit toutes les solutions possibles correspondant à la chaîne d'entrée.

Exemples d'étiquettes obtenues après analyse OPALE sur le texte "les avions rétropédalent ...":

- le mot "les" : une solution simple

  ⇒E1 = cat(artd), num(plu), ul(le)

- le mot "avions" : deux solutions simples

  ⇒E2 = cat(nmc), gnr(masc), num(plu), ul(avion)
  ⇒E3 = cat(vb), tmp(imp), per(4), ul(avoir)

- le mot "rétropédalent" : une solution complexe (multi-étiquette)

  ⇒E4 = cat(pref), ul(rétro) ET E5 = cat(vb), tmp(pres), per(6), ul(pédaler)

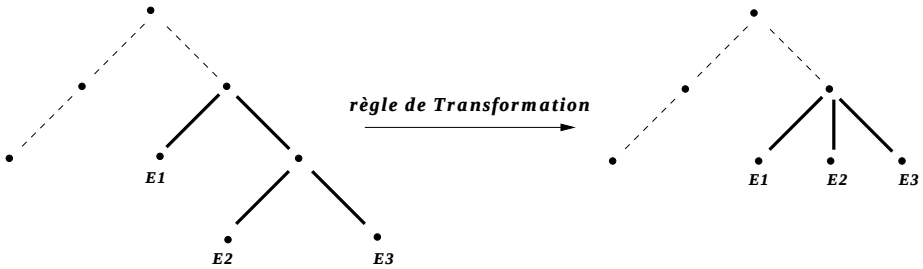
L'élément structuré résultant de l'analyse du texte "les avions rétropédalent ..."correspond à la figure 1.3.

1.3.2 Le langage TELESI

Le but du langage TELESI est de définir une transition entre des éléments structurés, plus précisément le langage TELESI permet de décrire des algorithmes de manipulations d'éléments structurés. Ces éléments structurés sont le résultat soit d'une analyse OPALE, soit de la lecture d'une structure, soit

d'un traitement TELESI antérieur. Ce langage effectue des remplacements de sous-structures (figure ci-dessous). Ce remplacement est décrit dans une règle de transformation. Ces règles de transformations sont regroupées dans des ensembles appelés grammaires élémentaires. Ces grammaires élémentaires ont un mode d'application qui décrit comment les transformations successives sont enchaînées, soit le mode itératif, soit le mode unitaire, soit le mode exhaustif. Ces grammaires sont elles-mêmes organisées en réseau. Une application complète d'une grammaire TELESI correspondra à un cheminement dans ce réseau de grammaires élémentaires. Ce cheminement est conditionnel et non déterministe, la solution considérée est toujours la première acceptée.

Exemple de remplacement d'une sous-structure :



1.3.3 Le langage AGATE

Le langage AGATE permet de définir une chaîne de caractères à partir d'une étiquette donnée. Ce système reçoit un ensemble d'étiquettes les unes à la suite des autres. Cette suite correspond au mot des feuilles de l'arborescence. Le but du langage AGATE est de définir une transition entre une multi-étiquette associée à un point d'un élément structuré et une chaîne de caractères. Le transducteur sous-jacent du langage AGATE est, comme pour le langage OPALE, un transducteur d'états finis non déterministe. La solution fournie par le langage AGATE est la première possible dans la recherche générale non-déterministe.

Chapitre 2

Les grammaires transformationnelles

Avant d'aborder le problème du transfert automatique des applications linguistiques d'ARIANE vers SYGMART, il était nécessaire de bien comprendre le fonctionnement des deux systèmes. Compte tenu du fait qu'ARIANE est le système actuellement utilisé au GETA, toutes les explications nécessaires à sa compréhension pouvaient nous être fournies directement par des membres de l'équipe du laboratoire. D'autre part, les similitudes entre ARIANE et SYGMART pouvaient nous permettre de n'étudier de manière approfondie qu'un des deux systèmes. Ces considérations font que nous nous sommes limités à l'étude du système SYGMART, et plus particulièrement à celle du sous-système TELESI, qui est l'élément principal de cet outil de TAO.

Dans un premier temps, nous allons brièvement résumer notre apprentissage du langage TELESI, en insistant surtout sur quelques éléments importants du système. Nous présenterons ensuite les différences essentielles entre TELESI et ROBRA.

2.1 Le langage TELESI

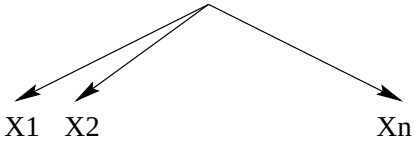
Le langage TELESI permet de décrire des algorithmes de transformations d'éléments structurés. Ces transformations sont en fait des remplacements de sous-arborescences, décrits par des règles de transformation. Un ensemble de règles de transformation constitue une grammaire élémentaire. Les grammaires élémentaires sont elles-mêmes organisées en réseau. Une application TELESI sur un élément structuré est définie par un cheminement conditionnel et déterministe dans ce réseau. C'est donc un enchaînement de transformations successives de l'élément structuré.

Les règles de transformations constituent l'élément de base du système. Elles se composent de deux parties : en partie gauche, le schéma de reconnaissance et en partie droite, le schéma de transformation.

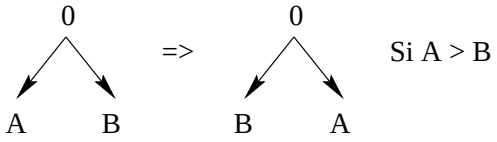
2.1.1 Un exemple : le tri à bulle

Comprendre le fonctionnement d'une application TELESI a nécessité que l'on travaille sur des exemples. Nous nous sommes donc intéressés à l'écriture de différents algorithmes de tri, dont un particulièrement simple, le tri à bulle.

A titre de rappel, les éléments manipulés par TELESI (les éléments structurés) sont des arbres étiquetés. Pour simplifier, nous pouvons dans un premier temps considérer que l'élément structuré d'entrée est un arbre plat constitué de la liste des éléments à trier. Les nombres à trier sont contenus par les étiquettes associées aux noeuds :



L'unique règle nécessaire pour réaliser le tri est la suivante (tri par ordre croissant) :



Cette règle est appliquée de façon itérative. L'arrêt du programme est obtenu lorsque la règle n'est plus applicable.

La syntaxe de la règle est la suivante :

```
&GRAM : BUBBLE( I ).
R_ECH : 0 ( 1 , * , 2 ) // N ( 1 ) > N ( 2 ) => 0 ( 2 , 1 )
```

où BUBBLE est le nom de la grammaire et R_ECH celui de la règle d'échange. Le "I" suivant BUBBLE est le mode d'application de la grammaire (itératif dans notre cas, c'est-à-dire que la règle sera appliquée autant de fois que possible). Le schéma de reconnaissance de la règle est :

```
0 ( 1 , * , 2 ) // N ( 1 ) > N ( 2 )
```

Celui de transformation est :

```
0 ( 2 , 1 )
```

2.1.2 Écriture des règles TELESI

Comprendre la syntaxe des règles de transformation nécessite quelques précisions supplémentaires.

2.1.2.1 Le schéma de reconnaissance

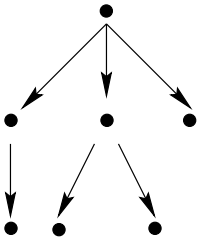
Un schéma de reconnaissance est une description d'arborescence complétée par deux ensembles de conditions (éventuellement vides) portant sur les étiquettes associées aux noeuds de l'arborescence. Cette description se fait donc en trois parties : une partie description structurelle et deux parties conditionnelles. La règle d'écriture est la suivante :

partie structurelle / partie conditionnelle propre / partie conditionnelle inter-sommets.

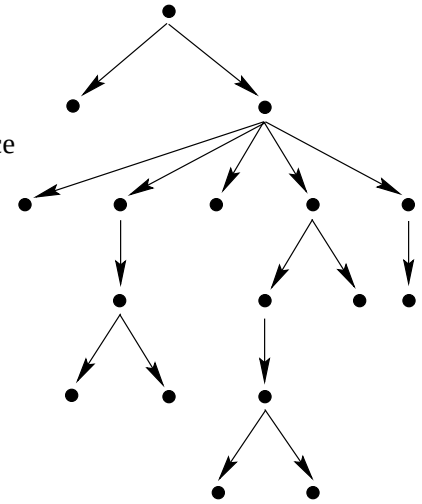
La description structurelle

Une sous-arborescence est un ensemble de points d'un élément structuré formant eux-même une arborescence.

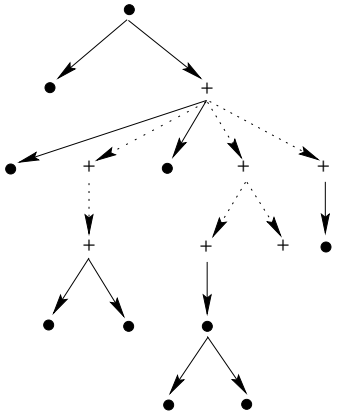
Ainsi, l'arborescence:



est une sous-arborescence de l'arborescence:



La décomposition s'effectue ainsi :

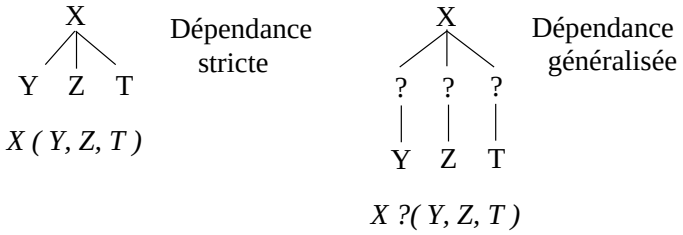


La partie structurelle est une description de sous-arborescence sur laquelle

s'applique certaines *contraintes*. Les différentes contraintes possibles sont :

– Contrainte de dépendance. Symboles : **? ( )**

Il existe deux sortes de dépendances. La dépendance classique(ou dépendance stricte) entre deux noeuds d'une arborescence, et la dépendance généralisée, introduite par un point d'interrogation. Dans l'exemple du tri à bulle, nous n'utilisons que la dépendance stricte.



– Contrainte de continuité. Symbole : *****

Cette contrainte permet d'interdire la présence de noeud entre deux frères d'une arborescence. Nous l'utilisons dans la règle R_ECH pour nous assurer que les noeuds 1 et 2 sont bien des frères directs (pas d'autre noeud entre eux).

– Contrainte de présence. symbole : %

Cette contrainte rend la présence d'un noeud facultative pour la présence d'un schéma dans une arborescence donnée.

– Contrainte d'ordre. Symboles : **- ; ,**

La contrainte d'ordre la plus utilisée est celle introduite par le symbole ",". Elle signifie que l'ordre spécifié dans le schéma de reconnaissance doit être respecté dans l'arborescence pour que le schéma soit présent.

Parties conditionnelles

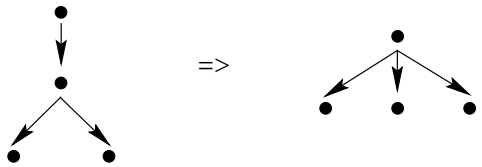
Les parties conditionnelles permettent de définir des conditions supplémentaires à la reconnaissance d'un schéma dans un élément structuré. On distingue deux types de conditions : les conditions propres, qui portent sur une étiquette associée à un noeud et les conditions inter-sommets qui portent sur des étiquettes associées à différents noeuds du schéma.

Ainsi, dans l'exemple du tri à bulle, "// N (1) > N (2)", définit une condition inter-sommets sur les valeurs des variables N associées aux noeuds 1 et 2. Il n'y a pas de condition propre.

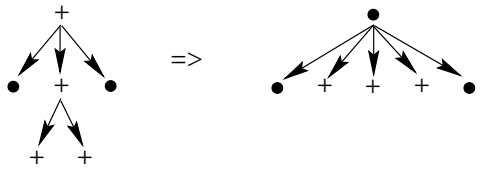
2.1.2.2 Le schéma de transformation

Une transformation d'arborescence est décrite par l'arborescence qui devra remplacer la sous-arborescence reconnue par le schéma de reconnaissance.
Exemple :

Soit la transformation :



Cette transformation modifie l'arborescence suivante par remplacement de la sous-arborescence repérée par des symboles "+".



La description d'un schéma de transformation suit les mêmes principes que ceux utilisés pour la définition d'un schéma de reconnaissance. La règle d'écriture est la suivante :

partie structurelle / partie modification propre / partie modification inter-étiquettes.

Description structurelle

La partie structurelle décrit une arborescence à laquelle s'ajoutent des *définitions* structurelles supplémentaires. Les différentes définitions possibles sont :

- Définition de dépendance. Symbole **( )**

La dépendance stricte est la même que pour la description du schéma de reconnaissance. En revanche, la dépendance généralisée n'existe plus.

- Définition de présence. Symbole **%**

Cette définition permet de faire dépendre la présence d'un point dans le schéma de transformation de la présence d'un autre point dans le schéma de reconnaissance.

- Définition de liste. Forme *** < , > ***

Une liste est définie par trois éléments : le père, le frère gauche et le frère droit de la liste.

Modifications d'étiquettes

Les parties modifications d'étiquettes permettent de préciser l'étiquette à associer à chaque point du schéma de transformation. L'étiquette à associer à un point peut être une étiquette déjà existante associée à un point du schéma de reconnaissance ou la définition d'une nouvelle étiquette.

2.1.2.3 Transfert automatique des listes

Un des aspects le plus important de TELESI est le transfert automatique des listes.

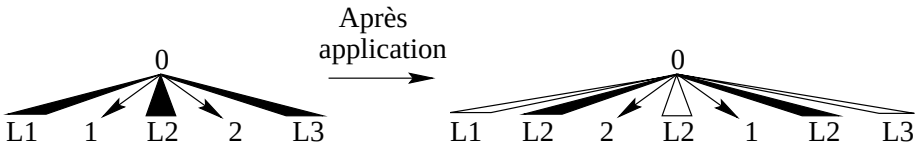
Pour qu'un noeud apparaissant dans le schéma de reconnaissance appartienne à l'arborescence finale, il faut qu'il soit explicitement replacé dans le schéma de transformation. Un noeud explicitement nommé dans schéma de reconnaissance qui n'apparaît pas dans le schéma de transformation est simplement supprimé.

En revanche, tout ce qui n'est pas explicitement nommé est implicitement transféré. Ce transfert automatique s'applique aux fils et aux frères des noeuds explicitement transférés.

Reprenons l'exemple du tri à bulle en modifiant la règle R_ECH (suppression de la contrainte de continuité entre 1 et 2) :

R_ECH : 0 (1 , 2) // N (1) > N (2) => 0 (2 , 1)

Le schéma de la règle (sur lequel nous avons fait apparaître les listes) est le suivant :



On constate que la liste L2 qui était entre les noeuds 1 et 2 à l'origine apparaît trois fois dans l'arborescence finale. La cause du problème est ici le transfert automatique des listes. En effet, les noeuds 1 et 2 conservent leurs fils et leurs frères. Dans le schéma de reconnaissance, 1 a pour frère gauche L1 et pour frères droits < L2 , 2 , L3 >. 2 est explicitement nommé dans le schéma de reconnaissance et est donc supprimé des listes de transfert implicite. Dans l'arborescence finale, il faut donc que 1 ait pour frères droits la liste < L2 , L3 >. D'où la recopie de la liste L2 à droite du noeud 1 dans le schéma de droite. Le noeud 2 conserve lui aussi ses frères, c'est pourquoi L2 apparaît à sa gauche.

2.1.3 Fonctionnement des grammaires TELESI

Les règles TELESI dépourvues de contrôle sur les modalités d'appel permettent d'écrire des applications simples. Mais une application complète de TAO nécessite un contrôle plus important sur l'appel des règles et sur l'élément structuré. TELESI permet de tels contrôles, que nous allons simplement énoncés ici sans entrer dans les détails.

2.1.3.1 Contrôle et récurrence de règle

Parmis l'ensemble des règles d'une grammaire, il est possible de définir un sous-ensemble de règles actives. Ces règles actives seront les seules à être testées

et éventuellement appliquées. Ce sous-ensemble de règles peut être modifié après chaque application d'une règle. Ce contrôle des règles actives comporte trois cas :

- Pas de modification de l'ensemble des règles actives (contrôle par défaut)
- Seule la règle appliquée reste éventuellement active.
- Seules les règles d'une liste déterminée restent actives

Ce contrôle des règles actives s'accompagne d'un contrôle sur l'élément structuré qui permet d'interdire l'application des règles à certains points du schéma.

Il existe également trois types d'appels récursifs qui peuvent porter sur :

- Un ensemble de règles.
- Une grammaire.
- Un réseau de grammaire.

Le contrôle de la récursion s'accompagne lui aussi d'un contrôle sur l'élément structuré. Les règles ou grammaires appelées par récurrence s'appliquent sur le résultat de la règle appelante.

2.1.3.2 Modes d'application des grammaires

TELESI permet également de contrôler le fonctionnement des règles d'une grammaire dans son ensemble. Ce contrôle s'effectue par le mode d'application d'une grammaire.

Les différents modes d'application d'une grammaire possibles sont :

- Itératif. Notation I. (Exemple du tri à bulle) : les règles de la grammaire sont appliquées indéfiniment. L'arrêt est obtenu lorsque plus aucune règle ne peut s'appliquer.
- Unitaire. Notation U(n). Les règles de la grammaire sont appliquées au maximum n fois.
- Exhaustif. Notation E. La grammaire est appliquée de façon itérative mais chaque règle appliquée est supprimée de la liste des règles actives. Le nombre maximum d'itérations est donc celui du nombre de règles de la grammaire.

2.2 Comparaison ROBRA / TELESI

Après avoir compris le fonctionnement global du système SYGMART, nous avons commencé à étudier le problème du transfert d'applications ARIANE. Nous avons donc étudié point par point les différences entre TELESI et ROBRA. Nous avons rencontré deux types de différences principales. Les premières se situent au niveau de la sémantique locale des règles, c'est-à-dire à leur application en un point de l'élément structuré. Les secondes se situent au niveau du contrôle des grammaires et des règles, et concernent donc leurs applications à tout l'élément structuré. Voici les quatre différences qui illustrent le mieux ce

que nous avons pu constater :

	ROBRA	TELESI
Dépendance généralisée	Au niveau des fils	Au niveau du père
Gestion des listes	Possibilité de nommer une liste dans le schéma de reconnaissance	Listes uniquement dans le schéma de transformation
Contrôle de la grammaire	Contrôle intuitif sur les règles et sur l'élément structuré (7 modes d'application). S'adresse plus à des linguistes	Contrôle plus complexe et moins complet. Correspond plus à une approche informatique du problème.
Recherche des schémas	Mode par défaut : de bas en haut. Possibilité de paramétrer le mode	De haut en bas. Non paramétrable

Tableau : principales différences entre ROBRA et TELESI.

Les problèmes de dépendance généralisée et de gestion des listes font partie des différences de sémantique locale, ceux concernant le contrôle de la grammaire et la recherche des schémas font partie des différences globales.

Effectuer un transfert automatique des règles de grammaires n'était pas envisageable sans trouver de solutions aux problèmes posés par ces différences de langages. Nous nous sommes dans un premier temps attaché à résoudre les différences de sémantique locale. Nous objectif était d'obtenir, pour une application d'une règle en un point donné d'une arborescence, une transformation identique. C'est uniquement dans un second temps que nous avons cherché à résoudre les problèmes d'applications globales sur l'élément structuré

Chapitre 3

Le transfert d'applications

Nous présentons ici les solutions que nous avons apportées aux problèmes liés aux différences des langages ROBRA et TELESI. Nous avons pour cela travaillé sur une application ARIANE que nous avons manuellement transférée vers SYGMART. L'application choisie est un système de génération du français développé actuellement pour le projet UNL. Nous avons travaillé du particulier au général, c'est-à-dire que nous avons commencé par essayer d'obtenir une équivalence sémantique locale des règles, avant de nous intéresser à la question des modes d'applications de ces règles.

Les différents éléments à convertir pour le transfert d'une application sont :

- Les fichiers de définition des variables.
- Les fichiers de formats.
- Les grammaires.
- Les dictionnaires.

Les seules difficultés se trouvent au niveau du transfert des règles, c'est donc uniquement sur cette partie que nous nous sommes attardés.

3.1 Transfert de règles

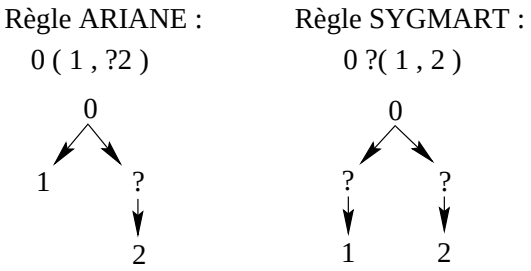
Les principaux problèmes que nous avons du résoudre sont les suivants :

- Gestion des dépendances.
- Gestion des listes.
- Contrôle des grammaires.
- Recherche des schémas.

Le problème de la gestion des dépendances est soluble efficacement. Quant aux problèmes liés à la gestion des listes, ils posent plus de difficultés au niveau de l'automatisation. Les problèmes de contrôle des grammaires et de recherche des schémas sont plus délicats et nécessitent des solutions algorithmiques complexes. Nous donnons ici un peu plus en détail les problèmes et leurs solutions :

3.1.1 Gestion des dépendances

La difficulté vient du fait que ROBRA gère la dépendance généralisée au niveau des fils, alors que TELESI le fait au niveau du père.
Par exemple :



La solution passe par l'utilisation d'une fonction prédéfinie de TELESI, *PROF* qui permet de consulter la profondeur d'un noeud dans l'élément structuré. La règle TELESI équivalente à la règle ROBRA est alors :

0 ?(1 , 2) // %PROF(1) = %PROF(0) + 1

C'est une façon de forcer le noeud 1 à être en dépendance stricte avec 0.

3.1.2 Gestion des listes

ROBRA permet de nommer explicitement une liste dans le schéma de reconnaissance. Cette possibilité pose des problèmes lors du transfert vers TELESI. Il est effectivement possible d'écrire des règles comme celle ci-dessous en ROBRA :

0 (1 ($X)) => 0 (1 (N , $X))

où $X dénote une liste (syntaxe ROBRA) et N un nouveau noeud. La règle SYGMART :

0 (1) => 0 (1 (G , *1*))

où *1* est la liste des fils du noeud 1, n'est pas équivalente. La raison vient du transfert automatique des listes. Après application de la règle TELESI ci-dessus, l'arborescence résultat contient deux fois la liste des fils de 1. Elle est dupliquée implicitement et ajoutée une seconde fois de façon explicite. Pour qu'il n'y ait pas duplication, il faudrait pouvoir la faire apparaître dans le schéma de reconnaissance, ce qui n'est pas possible.

Pour éviter le transfert automatique des listes, la seule solution consiste à ne pas réutiliser les noeuds du schéma de reconnaissance. Il faut alors créer de nouveaux noeuds, effectuer la recopie des étiquettes et faire un transfert explicite des listes.

La règle TELESI équivalente à celle de ROBRA est :

R_ECH : 0 (1) => 0 (N (G , *1*)) / N : 1.

N est le noeud créé en remplacement de 1 dans le schéma de transformation. N : 1 signifie que l'on associe à N l'étiquette du noeud 1.

3.1.3 Contrôle des grammaires

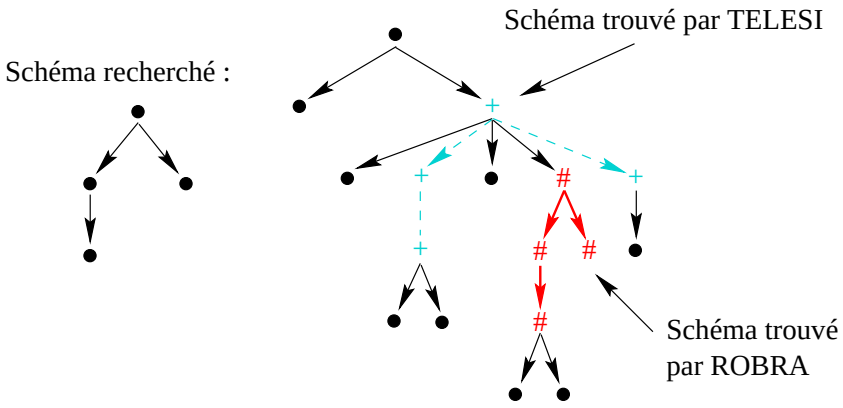
ROBRA permet de définir simplement des contrôles sur:

- La grammaire.
- Les règles.
- L'élément structuré.
- La recherche des schémas.

TELESI permet des contrôles similaires sur les trois premiers points. L'utilisation de ces contrôles est cependant plus complexe que sous ROBRA. Nous n'aborderons cependant pas ces problèmes ici.

Il n'existe aucun moyen de contrôler le mode de recherche des schémas sous TELESI. C'est certainement l'obstacle le plus important au transfert des règles ROBRA vers TELESI. Par défaut, sous ROBRA, les recherches se font de bas en haut et de gauche à droite. Sous TELESI, elles sont effectuées de haut en bas et de droite à gauche. Il en résulte que pour une arborescence donnée, dans laquelle un schéma de reconnaissance est présent plus d'une fois, les deux systèmes ne trouveront pas la même solution. Après application de la règle de transformation, l'arborescence résultat sera donc différente.

Par exemple :



Il a donc été nécessaire de simuler algorithmiquement une recherche de bas en haut sous SYGMART.

3.1.4 Recherche des schémas

Pour imposer un parcours de recherche de schéma de bas en haut, on modifie la grammaire et la règle de façon à obtenir l'algorithme suivant :

Initialisation : Marquage de tous les noeuds non feuilles de l'arborescence comme bloqués.

Boucle : tant que schéma de reconnaissance non trouvé

* Recherche du schéma de reconnaissance sur tous les noeuds non bloqués.

* Marquage des noeuds non bloqués comme traités.

* Déblocage des noeuds dont tous les fils ont été traités.

L'application de cet algorithme sur la figure précédente conduit à la bonne solution en douze itérations. Les modifications apportées à la grammaire initiale sont les suivantes :

- Ajout (avant l'appel de la grammaire) de la règle d'initialisation.

- Modification du mode d'application de la grammaire pour qu'elle devienne récursive.

- Modification de la règle initiale pour que le schéma ne soit recherché que sur les noeuds non bloqués.

- Ajout des deux règles qui marquent les noeuds comme traités et qui débloquent ceux dont tous les fils sont traités.

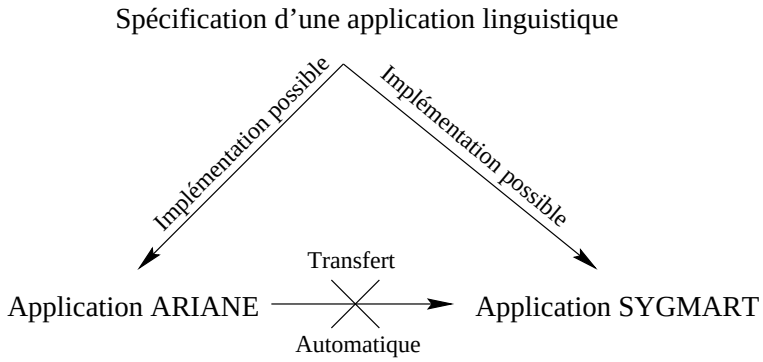
3.2 Analyse des résultats

Nous avons montré que le transfert automatique des applications ARIANE vers SYGMART est possible. La plupart des difficultés liées aux différences entre ces systèmes sont solubles de manière efficace. En revanche, le problème lié au mode de recherche des schémas multiplie le nombre de règles. Pour chaque règle pour laquelle il faut faire un parcours de bas en haut, nous ajoutons trois règles supplémentaires et nous introduisons une récursivité. Le mode de recherche par défaut de ROBRA est le mode de bas en haut. Nous n'avons par ailleurs jamais vu de règle pour laquelle le mode de recherche a été modifié. Il en découle qu'il faudra appliquer notre algorithme à la quasi totalité des règles. On arrive donc à une augmentation du nombre de règles qui accroit la complexité de l'application et nuit à son efficacité.

Nous n'avons pas pu mesurer à quel point il y a effectivement une dégradation des performances. Nous n'avons en effet pas terminé le transfert complet de l'application et nous n'avons donc pas pu la tester sur un exemple, ni mesurer son efficacité. Cependant, sur le schéma d'exemple précédent, nous obtenions douze applications de la grammaire. En supposant que la règle initiale ne contienne qu'une règle et compte tenu du fait que l'on ajoute trois règles dans la grammaire par règles existantes, le nombre de règles appliquées final est de quarante huit (quatre fois douze). Il n'y en a qu'une dans l'application initiale sous ARIANE. On peut donc raisonnablement considérer que l'efficacité de l'application est fortement dégradée.

Il ne faut cependant pas en déduire que SYGMART est moins performant qu'ARIANE. Ces systèmes ont la même puissance d'expression. Il est donc possible de développer des applications équivalentes sous les deux systèmes. Cependant, la conception même de ces applications diffère trop pour qu'il soit possible d'effectuer un transfert d'ARIANE vers SYGMART sans une perte importante de performance.

Le schéma suivant synthèse ces résultats :



Les deux systèmes sont effectivement comparables en terme de puissance d'expression, mais une application développée sous ARIANE pourra difficilement être transférée sous SYGMART sans en revoir toute la conception. Ceci est dû à l'utilisation durant le développement d'heuristiques non linguistiquement motivées et fortement dépendantes du mode de contrôle offert par le système.

Deuxième partie

Transfert du système de déconvertion d'UNL

Chapitre 4

Projet UNL

UNL est un projet de communication multilangage mis en oeuvre par l'Université des Nations Unies (UNU) basée à Tokyo. Ce langage permet une communication entre des personnes de langues différentes à l'aide de logiciels d'*enconversion* et de *déconversion*. UNL est un principe de codage du sens des textes (une sorte de HTML du contenu sémantique). La mise au point d'un tel langage, ainsi que des outils d'enconversion (langue X vers UNL) et de déconversion (UNL vers langue X) permettront le développement de nombreuses applications multilingues sur le réseau (recherche d'informations, veille technologique, traduction ...). Dans un premier temps, nous allons brièvement présenter le projet UNL. Ensuite, nous allons expliquer quels sont les états constitutifs des graphes UNL, et nous présenterons en détails le principe de construction d'un graphe. Puis nous montrerons quelques graphes réalisés à partir des textes HP avec les traductions obtenues après développement sur le système de déconversion utilisant ARIANE. Nous terminerons en concluant sur les différents problèmes existants.

4.1 Présentation du projet UNL

Le "langage universel de réseau" (UNL, Universal Networking Language) vise le codage du sens des documents, tout en respectant la diversité des traditions linguistiques. L'UNL, que dix-sept équipes de chercheurs ont adopté, est un "*langage-pivot*" débarassé de toutes les ambiguïtés des langues traditionnelles. Il peut ainsi servir d'intermédiaire à la compréhension entre les douzes langues actuellement déconverties. Le langage UNL permet de coder le sens de chaque énoncé d'un texte par un graphe.

4.2 Les graphes UNL

4.2.1 Qu'est-ce qu'un graphe UNL ?

Codé en UNL, un énoncé est un hypergraphe[6] où :

- Les arcs sont étiquetés par une relation sémantique (tels que agent, objet, but, etc.).
- Les noeuds sont étiquetés par un UW ("Universal Word", ou acception interlangue) et des attributs sémantiques.
- Chaque sous-graphe contient un noeud particulier appelé "entrée" (désigné par l'attribut sémantique .@entry).

4.2.1.1 Les relations sémantiques

Les relations sémantiques sont des relations binaires. Il existe environ une quarantaine de relations sémantiques.

Quelques relations sémantiques :

- **aoj**(attribut) définit une chose qui a un attribut.

  Syntaxe: aoj({state}, {thing})
  Exemples:

  aoj(red(icl>color), leaf(icl>thing)) ::: leaf is red / red leaf
  aoj(available(icl>characteristic), book(icl>thing)) ::: book is available / available book
  aoj(nice(icl>characteristic), ski(icl>event)) ::: skiing is nice

- **agt**(agent) définit une chose qui est à l'origine d'un évenement.

  Syntaxe: agt({event}, {thing})
  Exemples:

  agt(break(icl>event), John(icl>human)) ::: John breaks
  agt(save(icl>event), computer(icl>machine)) ::: computer saves

- **obj**(chose affectée) définit une chose qui est directement affectée par un évenement.

  Syntaxe: obj({event}, {thing})
  Exemples:

  obj(move(icl>event), table(icl>thing)) ::: table move / to move a table
  obj(think(icl>event), Mary(icl>human)) ::: think of Mary / a tought about Mary

4.2.1.2 Les UWs (Universal Words, ou acceptions interlangues)

Un UW désigne une collection d'acceptions interlangues (les différents sens du mot), bien que nous parlions souvent d'*un* sens du mot indiqué par un UW. L'anglais étant connu de tout les développeurs UNL, la syntaxe d'un UW est: **<mot anglais ou composé>(<liste de restrictions>)**, par exemple "look for(icl>action, agt>human, obj>thing)", dénote l'ensemble des sens de "look for" qui sont une action effectuée par un humain et ayant pour objet une chose.

4.2.1.3 Les attributs sémantiques

Les attributs sémantiques des UWs sont utilisés pour décrire le point de vue de la personne qui parle. Ceci inclut les phénomènes comme les actes de parole, les attitudes, "la valeur de vérité", etc. Ces attributs permettent d'enrichir la description avec plus d'informations sur comment la personne voit les événements et ses attitudes envers elles.

Quelques exemples :

- La personne pense que quelque chose est vrai, ou va le devenir : .@affirmative, .@inevitable, .@obligation, .@insistence
- La personne parle d'un événement achevé : .@complete
- La personne pense que quelque chose n'est pas vrai, ou ne va jamais devenir vrai : .@not, .@obligation-not
- La personne veut savoir si quelque chose est vrai : .@interrogative
- La personne parle d'un objet en particulier : .@def
- etc.

4.2.1.4 Syntaxe UNL

graphe ::= Liste de liens
lien ::= relation(UWa, UWa)
UWa ::= UW[attributs]*

4.2.2 Exemple de graphe UNL

Pour une meilleure explication sur les graphes UNL, nous allons construire un graphe à partir de la phrase : "*Ronaldo a marqué de la tête dans l'angle gauche du but.*"[3]. Une fois codée en UNL, la phrase devient une suite de mots universels : Ronaldo, score (*marquer*), goal (*but*), head (*tête*), corner (*coin*) et left (*gauche*). Chaque mot est complété d'indications destinées à lever toute ambiguïté. Il est précisé que *Ronaldo* est un humain. De même, il sera précisé que le but est une chose et non l'idée d'objectif; que la *tête* est une partie du corps et non une partie de quelque chose. Le verbe *marquer* reçoit toute une série de codes destinés à organiser la structure de la phrase : *Ronaldo* est son agent (le sujet dans notre grammaire), la *tête* est l'instrument (complément circonstanciel de manière), le *coin* est le lieu (complèment circonstanciel de lieu). De même, il sera précisé que l'adjectif *gauche* se rapporte au nom coin etc. Pour aider à transcrire la phrase en langage UNL, on peut s'aider d'un questionnaire visant à établir les relations entre les mots (figure 4.1) :

- Qui a fait quoi ? - Ronaldo a marqué
- Marqué quoi ? - le but
- Avec quel instrument ? - sa tête

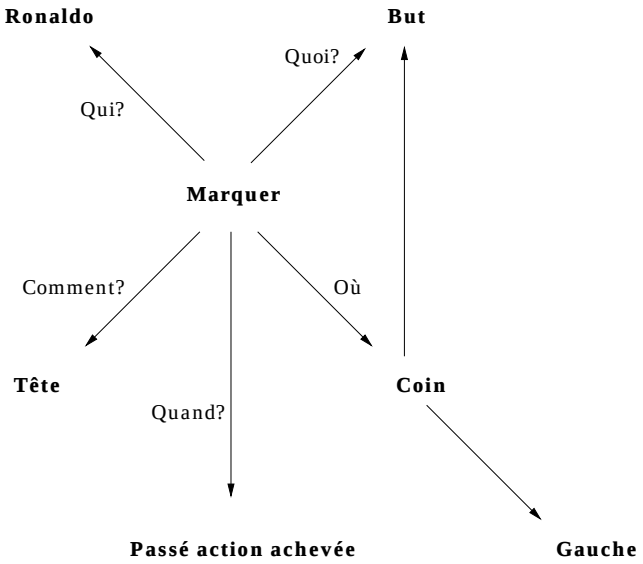


FIG. 4.1: Relations entre les mots de la phrase: "*Ronaldo a marqué de la tête dans l'angle gauche du but*"

- Marqué à quel endroit ? - le coin
- Le coin de quoi ? - celui du but
- Quel coin ? - gauche

Une fois convertie, la phrase "*Ronaldo a marqué de la tête dans le coin gauche du but*" devient :

<UNL>
agt(score(icl>event, agt>human, fld>sport).@entry.@past.@complete,Ronaldo(icl>human))
obj(score(icl>event, agt>human, fld>sport).@entry.@past.@complete,goal(icl>thing))
ins(score(icl>event, agt>human, fld>sport).@entry.@past.@complete,head(icl>body))
plt(score(icl>event, agt>human, fld>sport).@entry.@past.@complete,corner)
obj(corner,goal(icl>thing))
mod(corner,left)
</UNL>

En anglais, on aura: "*Ronaldo has headed the ball into the left corner of the net*"

4.2.3 Application de UNL à des textes Hewlett Packard

Hewlett Packard s'intéresse à la traduction automatique et plus particulièrement à UNL. Le service concerné a collaboré avec le GETA durant l'été pour

déterminer si UNL est adapté aux textes informatiques. Les textes de Hewlett Packard (Annexe 1 page 51) sont des textes anglais spécialisés dans le domaine informatique, donc avec un vocabulaire et des expressions propres à ce domaine. Les phrases constituant ces textes sont souvent très longues et très complexes. Voici quelques graphes caractéristiques tirés des fichiers HP avec leurs traductions actuelles obtenues à partir du sytème de déconversion utilisant ARIANE et les problèmes restant à traiter.

- **"Services Available."**

  aoj(available(icl>characteristic),service(icl>thing).@entry.@title.@pl)

  traduction obtenue : "Services disponibles."

- **"Services Available To All Registered Users."**

  aoj(available(icl>event), service(icl>thing).@entry.@title.@pl)
  gol(available(icl>event), user(icl>human).@pl)
  qua(user(icl>human).@pl, all(icl>quantity))
  obj(register.@pred, user(icl>human).@pl)

  traduction obtenue : "Services disponibles pour des tous usagers qui sont inscrits."
  problèmes : "pour des tous usagers" devrait être "pour tous les usagers". Ce problème est un problème de grammaire qui est assez facilement modifiable, nous pouvons le régler dès que sera validé le portage du système.

- ***"About cookies and the computer"***

  smd(about.@entry, :01)
  and:01(cookie(icl>computer_science).@pl, computer.@def)

  traduction obtenue : "<à_propos> des <témoin_de_connection> et l'ordinateur"
  La traduction est presque bonne mais il reste néanmoins deux problèmes à résoudre. Le premier est un problème de grammaire, "et l'ordinateur" devrait être "et de l'odinateur". Le second provient du système, les mots composés entourés de <> sont des locutions trouvées par le dictionnaire de transfert. Mais ces locutions ne sont pas traitées par le système. En effet, le système effectue un transfert de chaîne vers chaîne. La solution consisterait à coder le mot composé en un sous-graphe correspondant à celui-ci (voir section 5.2.1 page 39).

- **"But to use the computer, your browser must accept cookies, at least temporarily."**

  obj(use(icl>event).@pred, computer.@def)
  pur(accept(icl>event).@entry.@pred.@present.@obligation, use(icl>event).@pred)
  agt(accept(icl>event).@entry.@pred.@present.@obligation, browser)

obj(accept(icl>event).@entry.@pred.@present.@obligation, cookie.@def.@pl)
tmf(accept(icl>event).@entry.@pred.@present.@obligation, temporarily)
man(temporarily, at_least)
pos(browser, you.@pl)

traduction obtenue : "Votre navigateur doit accepter les <témoin_de_connection> pour utiliser l'ordinateur <provisoirement> <au_moins>"

La traduction est compréhensible mais il reste les problèmes liés aux locutions, ainsi qu'à l'ordre des mots.

- **"To provide a rich, powerful, and personalized user interface, Web applications like the computer often need to remember, across multiple screens, information which was only entered (or implied) once on previous screens."**

  traduction obtenue : "Des applications l'ordinateur <tel_que> un <web> sont un besoin souvent que se <souvenir_de_se> un <renseignements> à une <interface> un usager riche puissant? <<personalize>> est fournie" (voir Annexe 2 page 52 pour voir le graphe)

  La traduction n'est pas bonne. Le problème est lié à la complexité de la phrase.

4.2.4 Conclusion

Les textes de Hewlett Packard sont traduisibles en graphes UNL, mais de nombreux problèmes se sont posés :

- Les spécifications d'UNL sont très dures à comprendre et à utiliser;
- Les textes de Hewlett Packard sont très différents des corpus utilisés jusqu'alors pour le développement. Ils sont très complexes et très "lourds" (exemple trop de conjonction);
- Les locutions telles que "témoin de connection" sont très présentes dans les textes, et posent des problèmes lors de la phase de transfert.

Certains de ces problèmes sont liés au langage UNL lui-même (il n'est pas encore arrivé à maturité), d'autres sont dus à l'outil de déconversion encore en développement et inadapté à ce type de textes.

Afin d'obtenir une totale liberté dans l'adaptation nécessaire de déconvertisseur, nous avons choisi de travailler sur une version tournant sous le système SYGMART (et non plus ARIANE), plus flexible et plus rapide.

Chapitre 5

Système de déconversion

Dans le projet UNL, la stratégie pour "déconvertir" un hypergraphe UNL en une expression linguistique est libre. Chaque partenaire peut utiliser des outils communs ou des outils qui lui sont propres. Le déconvertisseur UNL-Français en cours de développement commence par une opération de "localisation" avec le format UNL, et ensuite effectue des phases classiques de transfert et de génération, utilisant l'environnement ARIANE-G5 et des outils spécifiques UNL, en particulier un convertisseur graphe vers arbre[4] .

Dans un premier temps, nous parlerons du système de déconversion français, puis nous décrirons les différents travaux réalisés sur les dictionnaires, adaptations et outils.

5.1 Présentation du système de déconversion

Le déconvertisseur UNL-Français utilise trois sortes de machines(figure 5.1), une pour chaque étape :

L'interface sur un serveur UNIX.

> Le script CGI est implémenté en perl sur un serveur UNIX. Il prépare les requêtes et les envoie via un socket (selon un protocole particulier) au Macintosh pour l'étape suivante. Il attend ensuite la réponse.

Des processus sur un serveur Macintosh (analyse de graphe et génération des arbres). Cette étape est constituée de trois phases :

> *L'analyse et validation du graphe.* Le texte UNL est parcouru par une grammaire LL1 pour construire le graphe correspondant. Si une erreur est rencontrée, un message est envoyé à l'utilisateur avec une explication de l'erreur. Le numéro de la ligne et de la colonne est retourné pour aider à localiser cette erreur.
>
> *Transfert lexical.* Les UWs sont traduites en français.

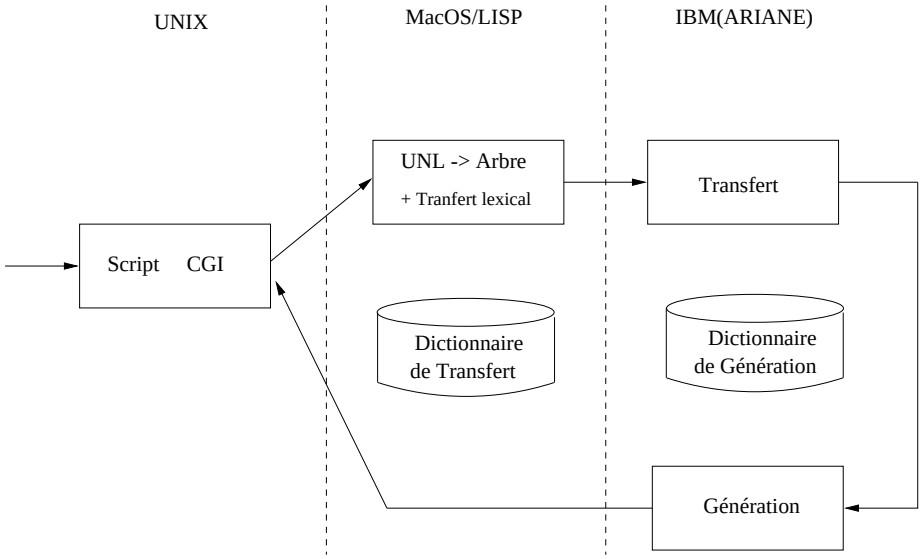


FIG. 5.1: Architecture logicielle du déconvertisseur UNL français

La génération de l'arbre. Le graphe obtenu dans la phase précédente est alors converti en un arbre pour être manipulable par les systèmes ARIANE ou SYGMART. L'arbre est écrit dans un message et envoyé à ARIANE-G5 (sur un mini-ordinateur IBM) pour traitement.

La génération du français sur un IBM.

Le système de traduction ARIANE fonctionne sur un IBM VM/CMS. Le traitement de l'arbre nécessite trois phases :

- Le transfert structurel de l'UNL vers le français;
- La génération structurelle vers la structure française;
- La génération morphologique vers le texte français.

5.2 Dictionnaires et outils

En plus des dictionnaires du système ARIANE (pour la génération du français), on utilise un dictionnaire de transfert : UNL vers français. Un dictionnaire UNL est constitué de trois colonnes : une colonne se rapportant à la traduction française, une autre constituée d'un ensemble de variables utilisé pour la déconversion par les outils linguistiques, et la dernière colonne se rapportant aux UWs avec ses rectrictions.

Exemples :

[bon]{CAT(CATADJ)}"good";
[utiliser] {AUX(AVOIR),CAT(CATV),VAL1(GN)}"use(icl>event)";
[ville]{CAT(CATN),GNR(FEM),N(NC)}"city(icl>area)";

5.2.1 Adaptation du serveur de déconversion

L'adaptation du serveur de déconversion a consisté en un premier temps à réaliser un dictionnaire spécialisé dans le domaine de l'informatique. Pour pouvoir traiter correctement les textes de HP. Nous avons introduit une nouvelle restriction pour les UWs : "computer_science".

Exemples :

[navigateur] {CAT(CATN),GNR(MAS),N(NC)} "browser(icl>computer_science)";
[session] {CAT(CATN),GNR(FEM),N(NC)} "login(icl>computer_science)";

Dans un second temps, le problème des locutions s'est posé. L'hypothèse était que le système ARIANE était capable de substituer des identifieurs par les arbres appropriés. Cependant, une telle solution cause de nombreux problèmes de gestion :

- Les dictionnaires doivent être gérés sous deux systèmes différents pour la même tâche, les dictionnaires UNL et les dictionnaires ARIANE;
- Quand les développeurs veulent modifier un dictionnaire, ils peuvent avoir à s'occuper du système ARIANE;
- Les dictionnaires doivent être synchronisés tout le temps.

Afin de réduire les problèmes de gestion, nous avons décidé de permettre aux lexicographes de spécifier directement les arbres linguistiques comme une traduction d'un UW. Mais nous devons suivre quelques contraintes, par exemple garder la compatibilité avec le format des précieux dictionnaires UNL.

Exemple :

[@@_1:'témoin'(3:'de',2:'connexion')]
{1":CAT(CATN),GNR(MAS),N(NC).2":CAT(CATN),GNR(FEM),N(NC).3":CAT(CATD).}
"cookie(icl>computer_science)";

5.2.2 Outils réalisé pour manipuler des dictionnaires

Nous avons mis au point des outils pour manipuler des dictionnaires. Ces outils ont été réalisés sous UNIX à l'aide des commandes *sed* et *sort*. La commande *sed* est un éditeur non intéractif. Cette commande permet d'appliquer un certain nombre de commandes sur un fichier puis d'en afficher le résultat (sans modification du fichier de départ) sur une sortie standard. Descriptions des outils (sources en Annexe 3 page 53) :

- *La fonction Igor :*

  Igor <Fichier_a_traiter> > <Fichier_Res1> <Fichier_Res2>

  Cette fonction permet de partager un dictionnaire en deux. Un Fichier_Res1 qui contient tous les UWs qui non pas encore été traités, et un Fichier_Res2 qui contient le reste.

- *La fonction Tri :*

  Tri <Option> <Fichier_a_traiter> <Fichier_Res>

  Cette fonction permet de faire un tri sur les colonnes des dictionnaires. L'option -1 permet de trier la première colonne du dictionnaire. L'option -2 (resp: -3) permet de trier la seconde (resp: troisième) colonne du dictionnaire et de la placer devant les deux autres colonnes.

  Ces outils très simples nous ont été utiles mais ont aussi été utilisés par l'équipe de développement linguistique d'UNL.

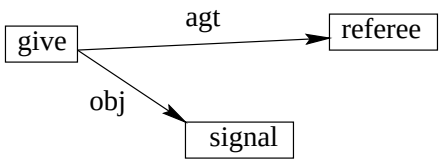
Chapitre 6

Le portage de l'application UNL

6.1 Le "noyau" ARIANE de l'application

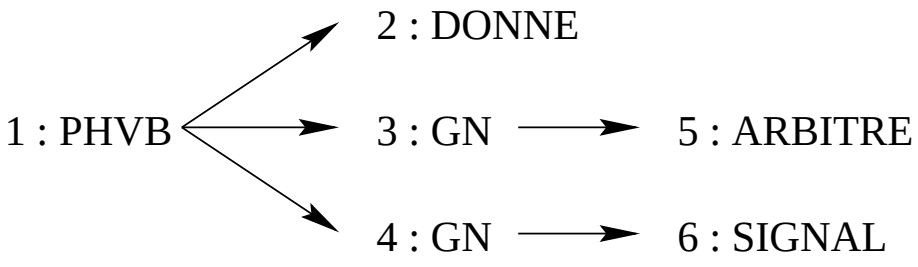
C'est uniquement sur la partie ARIANE que nous sommes intervenus. Elle comporte trois phases, la phase de transfert structurel (TS), la phase de génération syntaxique (GS) et la phase de génération morphologique (GM).

Nous détaillons le rôle des différentes phases à partir de l'exemple suivant : "L'arbitre donne un signal".



6.1.1 Le transfert structurel

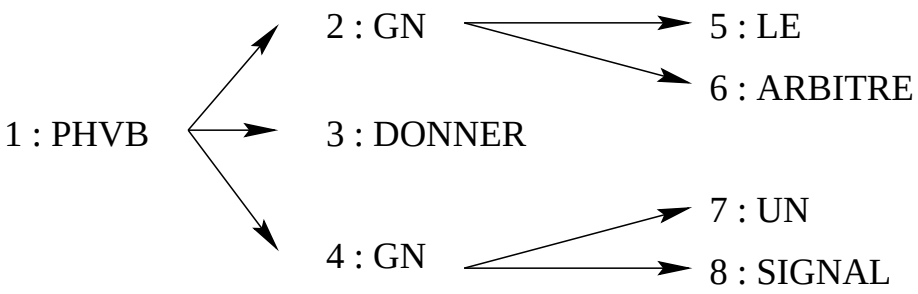
C'est un phase ROBRA. En entrée, elle prend un arbre qui est dérivé automatiquement d'un graphe UNL. Nous donnons ci-dessous l'arbre obtenu à partir de l'exemple de graphe précédent.



Dans cette phase, nous avons decidé de la stratégie d'expression du sens. La représentation choisie dans l'exemple est celle d'un phrase verbale, avec un verbre et deux groupes nominaux dont les fonctions syntaxiques sont déterminées par leurs étiquettes (pour le noeud 3, la variables FS vaut SUBJ -sujet- et pour le noeud 4 elle vaut OBJ1 -objet-).

6.1.2 La génération syntaxique

Le rôle de cette phase est d'introduire les éléments grammaticaux (article,...) nécessaires, de faire les accords, de mettre les mots dans le bon ordre, etc. Voici l'arbre obtenu en fin de phase :



C'est une phase ROBRA. Il n'y a donc pas de génération de texte, simplement des manipulations de la structure arborescente. Elle possède une grammaire imposante (plus de 28 pages...), mais pas de dictionnaire. Cette phase possède également des fichiers de macros, de formats d'affectations et de déclarations de variables. Les formats d'affectations ne sont guère différents des macros : ils permettent d'affecter tout un ensemble de variables d'un coup.

6.1.3 La génération morphologique

La génération morphologique est responsable de la transition de l'élément structuré vers le texte. Elle gère aussi la mise en forme du texte. La phrase obtenue pour l'exemple est :

L'arbitre donne un signal.

Un détail à remarquer ici est la transformation du "Le" en " L' ".

Cette phase est écrite en SYGMOR. C'est la dernière phase du processus de déconversion, mais c'est la première que nous avons portée, compte tenu du fait que les résultats étaient immédiats, ce qui simplifiait la mise au point. À l'opposé de la phase de génération syntaxique, elle se compose d'une grammaire assez réduite, contenant peu de règles. En revanche, elle possède cinq dictionnaires dont un particulièrement gros (plus de 30 000 entrées). Il y a également un fichier de macros et un de formats d'affectations. Bien sûr, il y a aussi l'indispensable fichier de déclaration de variables.

6.2 Les données de UNL sous ARIANE

6.2.1 Les variables

Voici un extrait du fichier de variables de la phase de génération morphologique.

```
                          -NEX-

$GNR    ==( FEM    ,MAS    ).

$MOD    ==( IND    ,SUB    ,CDL    ,IMP    ).

$P      ==( 1      ,2      ,3      ).

         ** Place de la ponctuation.
$PONC   ==( GCH    ,DRT    ).

$NUM    ==( SIN    ,PLU    ).
```

Le nom clé NEX introduit les déclarations de variables non-exclusives (qui peuvent prendre plusieurs valeurs simultanément). Beaucoup d'autres déclarations de variables sont possibles : les variables numériques, potentielles (dont on ne connaît pas à l'avance l'ensemble de définition), exclusives (c'est un peu un type énuméré), etc. Tous ces types de variables existent aussi sous SYGMART, il n'y a donc pas de difficulté autre que syntaxique. Les commentaires vont de la double étoile ("**") jusqu'au point suivant. Le reste constitue les déclarations. Le nom de la variable précède le "==" et la déclaration des valeurs possibles suit entre parenthèses. Les variables dont le nom est précédé d'un "$" sont importées de la phase précédente, mais on peut ne pas en tenir compte ici.

Le rôle des variables données en exemple est assez simple à comprendre. GNR signifie "genre". Cette variable permet de préciser si le mot associé au noeud est masculin ou féminin. MOD donne des informations sur le mode d'un verbe, P permet de spécifier la personne, etc.

Il faut noter qu'il existe sous ARIANE une variable particulière, qui n'apparaît pas dans les déclarations. C'est une variable de type chaîne, et la seule de ce type. Elle se nomme UL. Sous SYGMART, sa déclaration doit être explicite.

6.2.2 Les formats et les procédures

Voici un exemple de formats d'affectations. Ce sont en fait des macros qui permettent d'affecter plusieurs variables d'un seul coup.

```
DUIRE   01==term: -re,-sons,-t,-sit; --ex: clo,cui,tradui.
DUIRE   02   **FLXV-E-DUIRE.
EFUTUR  01==suffixe du futur en -ER.
EFUTUR  02   **DESVERB-E-EFUTUR.
ELS1    01==elision --ex: la,me,que,si.
ELS1    02   **ELIS-E-ELIS1.
```

Le nom du format, à gauche est suivi d'un numéro de ligne. Il est en effet possible de définir un format sur plusieurs lignes, à condition que son nom soit toujours le même, et que le nombre qui suit indique bien le numéro de lignes de la définition (ceci afin de permettre le tri des formats selon leur nom par un

éditeur de texte simple). Les commentaires sont compris entre le "==" et se terminent au premier point. La définition même du format est comprise entre "**" et le point final.

Note : les opérateurs sont entre '-'. Ainsi -E- est une affectation (égal).

Les procédures sont basées sur le même modèle, mais il en existe différents types. Les procédures d'affectations ne se différencient pas des formats, mais il existe aussi des procédures conditionnelles. Au lieu d'affecter directement plusieurs variables, elles permettent d'effectuer des tests. Certaines procédures possèdent une partie test et une affectation. Ce sont des procédures mixtes. Outre le fait que les formats ne permettent que des affectations, la différence avec les procédures est surtout liée au lieu d'appel.

6.2.3 Les dictionnaires

Les dictionnaires sont certainement, à cause de leurs tailles, les fichiers qu'il était le plus important de convertir automatiquement. L'extrait qui suit est tiré d'un des dictionnaires d'étiquettes utilisé pour la génération morphologique.

```
PART1                   ==FEMPLU  /        /ES,
                        ==FEMSIN  /        /E,
                        ==MASSIN  /        /,
                        ==MASPLU  /        /S,
                        ==        /        /?.
PART2                   ==FEMPLU  /        /ES,
                        ==FEMSIN  /        /E,
                        ==MAS     /        /,
                        ==        /        /?.
PRES1                   ==TA1     /EMUET   /E,
                        ==TA2     /EMUET   /ES,
                        ==TA3     /EMUET   /E,
                        ==TA6     /EMUET   /ENT,
                        ==        /EMUET   /E?.
```

Dans la première colonne (qui fait 24 caractères) se trouvent les entrées du dictionnaire. La seconde colonne est celle des conditions. C'est ici que l'on utilise les procédures conditionnelles. Comme la colonne suivante, celle des formats d'affectations, elle fait huit caractères au maximum (c'est aussi la taille maximale d'un nom de variable). La dernière colonne est la valeur affectée à la variable UL (nous rappelons qu'UL est la seule variable de type chaîne sous ARIANE). Sa taille maximale est de 26 caractères.

L'exemple donné ci-dessus est tiré du dictionnaire des terminaisons utilisé pour la génération morphologique. Ainsi, un mot dont l'étiquette associée contient les variables PGMV, égale à PART1, GNR, égale à FEM et NUM, égale à PLU sera généré avec la terminaison "es".

Une des particularités des fichiers d'ARIANE, qui n'apparaît vraiment qu'ici, c'est ce format colonne, qui rend l'analyse du fichier délicate avec les techniques actuelles (fondées sur la notion de séparateur).

6.2.4 Les grammaires

6.2.4.1 Grammaires SYGMOR

Les grammaires SYGMOR sont les plus simples. Ce ne sont que des enchaînements de règles. Il n'y a pas d'organisation en réseau de plusieurs grammaires.

```
INIT:    NIL ==ECD(1)/
         FLXV  (C):=FLXV(1);
         PGMV  (C):=PGMV(1);
         FLXN  (C):=FLXN(1);
         ALT   (C):=ALT(1);
         ELIS  (C):=ELIS(1)//
         (NOMADJ),(VERBE),(BONNE),(CONJG),
         (JETER),(CEDLEV),(ALGC), (MAJ),
         (NEG1),(UNION1),(UNION2),
         (ELS23),(ELS1),(PREF),(ELS4),(ELS5),(ELSF),FIN.

NOMADJ: FLXN(C)-NE-FLXNO
         == ECD(4) / //.

BONNE:  FLXN(C)-E-BONNE-ET-GNR(C)-E-MAS-OU-
        FLXN(C)-E-GRIS -ET-GNR(C)-E-MAS
         == / /
        TCHAINE(M,-2,"NN","N");
        TCHAINE(M,-2,"LL","L");
        TCHAINE(M,-2,"TT","T");
        TCHAINE(M,-2,"SS","S")/.
```

Le format d'une règle est le suivant :
Nom : condition == lecture d'un dictionnaire
/ affectations
/ appels de fonctions particulières (TCHAINE)
/ règles à appliquer après celle-ci
Tout est facultatif, c'est-à-dire qu'il peut ne pas y avoir de conditions (la règle s'applique tout le temps), pas de lecture du dictionnaire, etc.

6.2.4.2 Grammaires ROBRA

Les grammaires ROBRA sont divisées en deux parties. Dans la première, c'est l'enchaînement des grammaires entre elles qui est donné. C'est un peu le plan du réseau de grammaires. Pour chaque grammaire, le nom des règles qui la composent et son mode d'application (itératif ou pas, etc.) sont aussi donnés. Les règles sont détaillées dans la seconde partie.

Nous donnons en exemple d'abord le début du réseau de grammaire, puis une règle.

```
              -GRAM-

AMBIG(T): AMBK;          ** Copi{r la classe K sur l'homographe.

                         --> LOCUT.

LOCUT (EL):              ** Traitement des locutions.
```

```
LOCUT1A,        ** GN(Centre(Recherche(Espace)) ==
                   GN(Centre,GN(Recherche(Espace))).
LOCUT1B,        ** GN(Porte(Fene^tre))== GN(Porte,Fene^tre).
LOCUT2A,        ** GN(Pouvoir(Grand(Tr}s)) ==
                   GN(GA(Grand(Tr}s)),Pouvoir).
LOCUT2B,        ** GN(Forme(Plate))   == GN(Plate,Forme).
PPNOM1,         ** GN(Pr{sent(Jusque))== GN(Jusque,A,Pr{sent).
PPNOM2,         ** GN(Doute(Sans))    == GN(Sans,Doute).
PPNOM3,         ** REG et CATN(Prep)  == GN(Prep,CATN=GOV);
                   Ex: Cadre(Dans)    == <LOCF>(Dans,Cadre).
PPNOM4;         ** GN(Gra^ce,COMP)    == GADV(Gra^ce,ARG1).

                --> SBINF.
```

La grammaire d'entrée est la première, AMBIG. Elle ne contient qu'une seule règle, AMBK. La grammaire suivante dans le réseau sera LOCUT.

Voici maintenant la règle AMBK, qui n'est pas trop complexe.

```
AMBK:   0(1(2))/0:$KN0;1:$GOV;2:$HOMOGR-ET-$K0
        ==0(1(2))//2:2,K:=K(0)
```

La syntaxe est proche de celle de SYGMART (le lecteur peut revoir l'exemple du tri à bulle au chapitre 3).

6.3 Le portage

En dehors des grammaires, tout a pu être porté automatiquement. Nous avons fait un outil pour chacun des types de fichier à convertir en SYGMART. Au premier abord, nous avions pensé faire une analyse des fichiers avec Lex et Yacc. Il s'est avéré que c'était une erreur. En effet, le format colonné des fichiers ARIANE aurait nécessité une analyse elle aussi en colonne. Faire une analyse séquentielle s'est révélé particulièrement difficile pour les dictionnaires, par exemple, puisque tous les caractères spéciaux qui auraient pu servir de séparateurs étaient également utilisés dans certaines entrées des dictionnaires.

Nous avons malheureusement pris conscience de cette difficulté à un moment où nous avons préféré continuer sur notre lancée plutôt que de remettre tout notre travail en question. Tous les outils que nous avons créés sont donc en Lex et Yacc.

Pour les grammaires, le problème est tout autre. Nous avions montré qu'il est impossible de les porter automatiquement. Une simple traduction syntaxique est en effet insuffisante. Il faut réécrire les règles en conservant leur sémantique, ce que nous n'avons pu faire que manuellement.

Le volume total de l'application est énorme. Les dictionnaires contiennent plus de 30 000 entrées, ce qui représente 1,2 Mo de données au format texte. L'ensemble des grammaires des différentes phases totalise, quant à lui, pas loin de 40 pages. À l'heure actuelle, toutes les règles ont été portées sur SYGMART. L'application est donc opérationnelle, au sens où elle génère effectivement une traduction. Mais pour beaucoup de règles, nous n'avons effectué qu'une simple adaptation syntaxique. Il en résulte un certain nombre de bugs, et une mauvaise qualité de traduction. Il reste donc à retravailler certaines règles pour retrouver la sémantique d'origine (celle des règles ARIANE).

6.4 Résultats obtenus

Pour chacunes des phrases anglaises ci-dessous, nous avons mis le graphe correspondant et le résultat obtenu sous SYGMART :

- *"About cookies and the computer."*

  [S]
  smd(about.@entry, cookie(icl>computer_science).@pl.@def)
  and(cookie(icl>computer_science).@pl.@def, computer.@def)
  [/S]
  * .. A|1 PROPOS DES TE|1MOINS DE CONNEXION ET DE L' ORDINATEUR

  La déconvertion est bonne mais il reste encore un problème avec le placement du point. (A|1 est le codage de à, etc)

- *"He and I are very good friends."*

  [S]
  mod(friend(equ>pal).@entry.@pred.@pl,good) mod(good,very)
  aoj(friend(equ>pal).@entry.@pred.@pl,he) and(he,i)
  [/S]
  * LUI ET MOI SOMMES DE TRE|2S BONS AMIS ..

  La déconvertion marche parfaitement.

- *"But to use the computer, your browser must accept cookies, at least temporarily."*

  [S]
  obj(use(icl>event).@pred, computer.@def)
  pur(accept(icl>event).@entry.@pred.@present.@obligation, use(icl>event).@pred)
  agt(accept(icl>event).@entry.@pred.@present.@obligation, browser)
  obj(accept(icl>event).@entry.@pred.@present.@obligation, cookie.@def.@pl)
  tmf(accept(icl>event).@entry.@pred.@present.@obligation, temporarily) man(temporarily, at_least)
  [/S]
  * UN NAVIGATEUR DOIT? ACCEPTER PROVISOIREMENT AU MOINS UN TE|1MOIN DE CONNEXION POUR UTILISER L' ORDINATEUR ..

  Il reste un problème lié à une règle de conjugaison qui introduit le point d'interrogation malgrès une déconversion correcte.

Conclusion

Nous avons montré qu'il est possible de palier aux différences entre les systèmes ARIANE et SYGMART, et ce de façon systématique. Il n'y a donc pas d'obstacle technique au transfert des applications linguistiques entre les deux systèmes. Cependant, un tel transfert accroît démesurément la complexité de l'application. Il en résulte une perte d'efficacité telle qu'il n'est pas envisageable de réaliser ce transfert. Nous avons donc abandonné l'idée de la réalisation d'un transfert *automatique* des applications du système ARIANE vers le système SYGMART. En revanche, nous avons réalisé tous les outils nécessaires à un transfert semi-automatique.

Notre travail sur le système de déconvertion de l'UNL nous a permis d'obtenir une application qui génère du français à partir d'un graphe UNL, sous le système SYGMART. Mais ce succès est à relativiser : les résultats obtenus ne sont pas de très bonne qualité.

La plupart des problèmes rencontrés sont liés au langage UNL lui-même qui n'est pas encore arrivé à maturité. Durant ce stage, nous nous sommes surtout attachés à améliorer les résultats de déconversion française pour les textes HP. Les discussions sur le langage UNL ne pouvant avoir lieu qu'avec les 17 partenaires, cette tâche n'a pas été aisée, du fait des spécificités des textes techniques de HP (longues phrases, nombreuses locutions, sous-langage différent de celui étudié dans le projet UNL).

D'autres problèmes viennent de la phase de génération syntaxique (GS). C'est en effet celle sur laquelle nous avons fait le plus de travail "à la main". Le nombre de règles de grammaire de cette phase est énorme, et c'est malheureusement le seul point qu'il nous était impossible de porter automatiquement. Afin de gagner du temps, nous avons cherché à rester le plus proche possible (dans la forme) des règles ARIANE lorsque nous avons traduit cette grammaire pour SYGMART. Cela ne prête pas forcément à conséquence dans beaucoup de cas, mais certaines des règles ont désormais un comportement très différent de celui des règles originales. Nous sommes à l'heure actuelle en train de retravailler cette phase.

Enfin, il reste le problème de l'efficacité. Toujours parce que nous avons cherché à "coller" aux règles ARIANE, la structure globale de l'application est mal adaptée à SYGMART, d'où une importante perte d'efficacité. Il y a peu de chose à faire pour la GS, en revanche, il est tout à fait possible d'optimiser la GM. Par contre, cela nécessiterait de complètement repenser cette phase, par

exemple en la réécrivant en partie en TELESI plutôt qu'en AGATE. Idéalement, si l'on voulait s'en tenir à AGATE, il faudrait modifier le dictionnaire, c'est-à-dire plus de 30 000 entrées, ce qui n'est pas pensable. Enfin, il devrait être possible d'accroître l'efficacité de l'application en supprimant les phases de transition que nous avons artificiellement introduites pour reproduire le fonctionnement exact d'ARIANE. La seule raison d'être de ces phases vient des déclarations de variables : un fichier par phase pour ARIANE, qui implémente un mécanisme permettant de passer d'un jeu de variables à l'autre et qui n'existe pas sous SYGMART. Mais ce dernier devrait nous permettre de faire toutes les déclarations dans un même fichier, en regroupant les différents jeux de variables dans des "variables globales" (on peut voir les variables globales comme des structures en C). Avec seulement de petites adaptations dans les grammaires, pour que le compilateur sache à quelle variables globales il doit se référer, il devrait alors être possible de supprimer les phases transitoires.

Sur le plan personnel, ce stage nous a permis de découvrir un domaine de l'informatique que nous ne connaissions pas. Nous avons appris à programmer dans des environnements très différents de ceux dont nous avions l'habitude, avec des langages qui nécessitent une approche et une pensée particulière. Cette expérience nous a d'autre part confortée dans l'idée de poursuivre nos études et de nous orienter vers le monde de la recherche.

Bibliographie

[1] **Brunet-Manquat F. et Guyotot Ol. (Avril 1999)** *Etude du transfert automatique d'applications linguistiques du système ARIANE au système SYGMART.* rapport de maîtrise (TER, Travaux Etude et de Recherche).

[2] **Boitet Ch. et Sérasset G. (Avril 1999)** *UNL-French deconversion as transfert & generation from an interlingua with possible quality enhancement through offline human interaction.* MT-Summit, Singapore.

[3] **Delbecq D. (Décembre 1998)** *UNL, un nouvel Esperanto pour le web.* Le Monde interactif.

[4] **Sérasset G. (août 1999)** *Improving the Lexical Transfert in UNL-F system.* rapport interne du GETA

[5] **Sérasset G. (Décembre 1998)** *Report on UNL-F project, year 1998, contract A.* rapport interne du GETA

[6] **UNL Center** *The Universal Networking Language (UNL) Specifications.* rapport interne de l'UNL Center

[7] **UNU/IAS (The United Nations University/Institute of Advanced Studies, 1996)** *UNL Universal Networking Language: An Electronic Langage for Communication Understanding and Collaboration*

Annexe 1: Quelques textes HP

About Cookies and HP-stuff

If you have registered or logged-in, but are asked to login again immediately afterward, then your browser is probably rejecting cookies. Netscape Navigator 3.0, Microsoft Internet Explorer 3.0, and later versions allow users to reject cookies on a case-by-case basis, or always. But to use HP-stuff, your browser must accept cookies, at least temporarily.

More about Cookies and HP-stuff

For your information, this screen will explain more about what cookies are, and how HP-stuff uses them.

What is a cookie?

To provide a rich, powerful, and personalized user interface, Web applications like HP-stuff often need to remember, across multiple screens, information which was only entered (or implied) once on previous screens. For example, many screens in HP-stuff need to know your User ID, registered name, e-mail address, linked System Handles, and so forth. When you login with your User ID, you enter (or imply) all of this information at that time. But the subsequent screens you visit after logging-in need to know this information, too.

Annexe 2: Exemple de graphe HP

"To provide a rich, powerful, and personalized user interface, Web applications like the computer often need to remember, across multiple screens, information which was only entered (or implied) once on previous screens."

agt(need.@entry.@pred.@present, application.@pl)
man(need.@entry.@pred.@present, often)
obj(need.@entry.@pred.@present, remember.@pred)
rsn(need.@entry.@pred.@present, provide.@pred)
obj(provide.@pred, interface) mod(user, interface)
aoj(rich, interface)
and(powerful, rich)
and(personalize.@pred, powerful)
mod(Web, application.@pl)
smd(application.@pl, computer.@def)
bas(like(icl>comparaison), computer.@def)
obj(remember.@pred, information)
scn(remember.@pred, :03)
aoj:03(multiple, screen(icl>computer):01.@entry.@pl)
aoj(enter.@pred.@past, information)
man(enter.@pred.@past, once)
mod(only, once)
plc(enter.@pred.@past, screen(icl>computer):02.@pl)
aoj(previous, screen(icl>computer):02.@pl)
;— entre parenthese normalement ....
or(enter.@pred.@past, imply.@pred.@past)

Annexe 3: outils pour manipuler les dictionnaires

Source de la fonction Igor

```
# Recupere les lignes {}
sed -n "s/\{\}/\{\}/w $2" $1
# Recupere le reste
sed "/\{\}/d" $1 > $3
```

Source de la fonction Tri

```
# Standardisation du dictionnaire
sed "s/ //g" $2 > $2.sansesp
sed -n -e "s/]/] /" -e "s/\}/\} /w $2.avecesp" $2.sansesp
# Tri selon l'option demandée
case $1 in
                -1) sort -f -d $2.avecesp > $3;;
                -2) sort +1 -2 -f -d $2.avecesp > $3;;
                -3) sort +2 -3 -f -d $2.avecesp > $3.arev;
                    sed -e "s/\(.*\) \(.*\) \(.*\)/\3 \2 \1/" $3.arev
   > $3; \rm $3.arev;;
                *) echo "Option inconnue";;
esac
# Efface les fichiers temporaires
\rm $2.sansesp
\rm $2.avecesp
```