Dimi

Yann Régis-Gianas et François Yvon

GET/ENST ET CNRS, LTCI UMR 5141
46 rue Barrault - F-75013 Paris
Tél. : +33 (0)1 45 81 77 59 - Fax : +33 (0)1 45 81 31 19
Mél : {yvon,regis}@enst.fr

ABSTRACT

DIMI is a tool aimed at developing grapheme-to-phoneme (G2P) transcription systems. It provides the user with a declarative formalism for expressing the transduction of symbolic streams annotated with predicates. This formalism is organised in 3 layers : *contextual rules* using extended rational expressions express the individual rewriting rules. Rules are organized in *strategies*, which specify how individual rules should be applied : sequentially, in parallel... Strategies are further organised in *modules*, allowing the user to manage complex rule sets. A modular, complete rule set for French is also presented, which illustrates the various possibilities of this language.

1. INTRODUCTION

DIMI est un système conçu pour le développement de transducteurs graphème-phonème à base de règles. Par rapport aux nombreux systèmes existants (eg. [7, 6, 1]), DIMI se caractérise principalement par : une organisation modulaire du système de récriture, intégrant de manière intrinsèque une capacité à gérer des indéterminations ; un environnement complet de développement, intégrant un compilateur de règles et de dictionnaires et diverses fonctionnalités de traçage et de déboggage.

1.1. Transcription graphème-phonème

La transcription graphème-phonème est traditionnellement vue comme la récriture d'un flux graphémique en la séquence phonémique représentant l'oralisation de ce flux. Cette récriture est précédée d'un prétraitement du flux d'entrée incluant notamment : une normalisation orthographique, comportant un traitement des nombres, des abbréviations... ; un étiquetage morpho-syntaxique, visant à désambiguïser les homographes hétérophones ; une identification (grossière) des constituants syntaxiques, visant à localiser les principaux contextes de liaison (eg. [8]).

Dans les systèmes par règles, la transcription de ces flux "enrichis" est effectuée par application de règles de récriture contextuelles, le contexte intégrant à la fois le voisinage graphique des graphèmes et des informations de "plus haut" niveau (lexicales, morpho-syntaxiques...) accumulées lors des prétraitements. La sortie du système contient typiquement une transcription phonétique *unique*, organisée en un flux de phonèmes, éventuellement entremêlés de marqueurs nécessaires aux traitements ultérieurs, en particulier prosodiques : de fin de mots, de groupes...

Pour le français, de tels systèmes intègrent typiquement des centaines, voire des milliers de règles, qui entretiennent entre elles des dépendances parfois subtiles [3]. Ces règles doivent ainsi être extrinsèquement ordonnées, ce qui conduit parfois à des décisions de traduction prématurées, i.e. sans que l'intégralité du contexte de la règle soit connues. Cette configuration apparaît par exemple lorsque l'on considère la phonétisation des sigles ([2]) ; un cas peut-être plus problématique est l'application, par la majorité des systèmes, des règles d'élision du e-muet, à un moment du traitement où les informations prosodiques, en particulier concernant le rythme, n'ont pas encore été calculées.

Cette vision de la tâche de transcription, héritée des applications de TTS, doit évoluer pour mieux servir d'autres contextes applicatifs : les systèmes de reconnaissance vocale, pour lesquels il est important de savoir produire des variantes de prononciation ; la synthèse à partir de concepts, qui met à disposition du système de transcription des informations supplémentaires, de nature sémantique et/ou pragmatique, et en attend en retour une flexibilité accrue : par exemple une capacité à produire des transcriptions de registres variés, correspondant à des débits d'élocution plus ou moins rapides, plus ou moins formels, au rendus d'émotions, d'accents régionaux...

Commander la génération des variantes *cohérentes* pour des groupes de mots, voire pour des phrases, est toutefois difficile dans la mesure où :

- (i) on manque encore de descriptions qui permettraient d'exprimer par des règles l'impact sur la chaîne segmentale (sur les liaisons, le degré d'assimilation...) du choix de tel ou tel style d'élocution.
- (ii) gérer des variantes, donc du non-déterminisme dans la traduction, présuppose une structuration des bases de règles, qui en particulier permette de localiser les points d'indétermination et d'exprimer des règles de filtrage qui lèvent ces indéterminations.

1.2. Le système DIMI

DIMI est un environnement de développement de systèmes de transcription qui permet de décrire simplement et de gérer efficacement des indéterminations. DIMI s'appuie sur un langage de programmation dédié à l'expression de récritures dans des flux symboliques. DIMI est un DSL (*Domain Specific Language*) [5] et tire parti des fortes restrictions du langage pour fournir des opérateurs efficaces et bien adaptés à la spécification de systèmes de transcription, ou plus généralement de traducteurs.

DIMI contient un langage déclaratif spécifiant la traduc-

tion d'un ou plusieurs flux de symboles annotés par des prédicats. Il est organisé autour de trois couches : un *langage de règles* définissant les récritures locales, un *langage de stratégies* déclarant l'agencement des règles et un *système de modules* facilitant réutilisation et maintenance. DIMI inclut également un ensemble d'outils (compilateur, interpréteur, machine virtuelle) qui permettent l'écriture, la compilation et le test de modules. Un premier système complet de transcription du français a été développé pour tester DIMI dans un cadre linguistique concret. En revanche, DIMI n'inclut pas d'outils de prétraitement : les informations nécessaires à la mise en œuvre de la transcription doivent être calculées au préalable.

Cet article est organisé comme suit. La section 2 décrit le noyau du système, i.e. le formalisme des règles. La section 3 présente les outils pour combiner ces briques élémentaires dans un processus complet de traduction. Enfin, le jeu de règles du français est présenté dans la section 4.

2. RÈGLES

2.1. Interaction entre connaissance et traduction

Classiquement, la traduction graphème-phonème par règles consiste en la transformation progressive d'une chaîne orthographique en une chaîne phonétique. Certaines traductions complexes, qui résultent de l'interaction de nombreuses règles rendent nécessaire l'introduction dans le flux de "méta-caractères" qui servent à propager d'une règle à une autre des informations collectées localement. Dès lors, la chaîne est utilisée à la fois comme objet traduit et comme support d'une information nécessaire à la traduction. Ces méta-caractères doivent être pris en compte par la récriture, introduisant une complexité supplémentaire dans les règles.

Comme proposé, eg. par [4], une autre approche consiste à travailler sur une structure de données plus souple. Nous avons choisi ici de séparer de manière explicite la connaissance collectée sur le flux et le flux lui-même. Ainsi, nous définissons l'objet de la traduction sous la forme :

- d'un alphabet fini Σ ;
- de flux, chaque flux étant un mot de $\Sigma^\star$;
- d'un ensemble *extensible* de prédicats d'arité multiple sur des segments des flux, qui associent des noms avec des ensemble d'intervalles des flux.

Un objet de ce type est donné en exemple Figure 1.

Flux	l 0	a 1	m 2	E 3	R 4
Predicats	Syllable		Syllable		
	OpenSyllable		ClosedSyllable		
	Word		Word		

FIG. 1: Flux et prédicat
Au flux phonémique $/lamER/$ *sont associés les prédicats* `Syllable`*, qui porte sur les intervalles* $[0,1]$ *et* $[2,4]$ *;* `OpenSyllable`*, qui porte sur l'intervalle* $[0,1]$*...*

2.2. Formalisme des règles

Les règles de récritures contextuelles fournissent un outil puissant pour les systèmes de transcription [9]. Elles sont usuellement présentées sous la forme suivante :

$$\phi[a]\psi \rightarrow \lambda[b] : p$$

qui exprime la récriture du mot a en b dans un contexte décrit par des expressions rationnelles ϕ, ψ, et λ et des conditions supplémentaires p condition sur l'entrée. Ce formalisme vaut pour un parcours gauche-droite de l'entrée, impliquant que dans la sortie, le contexte gauche λ de b est disponible. Nous étendons ce formalisme selon :

$$\phi[a]\psi \rightarrow \lambda[b] : p \Rightarrow c$$

où c est une *conclusion de la règle* et où a et b sont des *expressions rationnelles étendues*, telles que le langage dénoté par a est non vide, et que le langage dénoté par b est fini. Le fait que la partie droite d'une règle puisse être un langage fini quelconque permet d'introduire des indéterminations dans les sorties. Ce point sera discuté plus en détail à la Section 3.2. D'un point de vue opérationnel, une règle est interprétée comme suit : si à la position courante du flux, il existe un sous-mot capturé[1] par a, précédé (resp. suivi) d'un segment capturé par ϕ (resp. ψ), si le suffixe de la sortie courante est capturé par λ et si p est vérifié, alors (i) on concatène *chaque mot de* b dans de nouvelles sorties et (ii) on *ajoute* la conclusion c à l'ensemble de leurs prédicats (les prédicats associés au segment de l'entrée sont donc intégralement recopiés dans la sortie).

Deux constructions interviennent donc dans les règles : les expressions rationnelles et les prédicats. La syntaxe des expressions rationnelles est définie Figure 2. En plus des opérateurs classiques (l'itération, la disjonction...), elle inclut un mécanisme de *filtrage de prédicat* qui permet de vérifier des contraintes arbitraires, y compris structurelles, sur des segmemts du flux. Il est également possible de nommer un segment pour y faire référence dans la condition. Le mécanisme de filtrage met en évidence la notion de segment (ou d'intervalle) étiqueté, qui est l'élément de base du langage. Ainsi, par exemple, l'expression `%Word{%Syllable* @x{%Syllable}}` permet de capturer un segment qui (i) se décompose en une séquence de sous-segments satisfaisant le prédicat `Syllable` (ii) satisfait lui-le prédicat `Word` ; et de nommer la dernière syllable x.

motif	::=	mot	*mot* $\in \Sigma^\star$
	|	'%' id ('{' motif '}')?	*filtrage de prédicat*
	|	'@' id ('{' pattern '}')?	*association de nom*
	|	motif '*'	*répétition arbitraire*
	|	motif '+'	*répétition non vide*
	|	motif '?'	*motif optionnel*
	|	motif '|' motif	*alternative*
	|	motif motif	*concaténation*
	|	'(' motif ')'	*motif parenthésé*

FIG. 2: Syntaxe des expressions rationnelles étendues

Les prédicats sont statiques ou dynamiques : ils correspondent soit à des annotations de segments, calculées dynamiquement par application de règles ; soit à des formules logiques du premier ordre dont les atomes sont des énumérations de symboles ou des noms de prédicats. En plus de la définition de prédicat, la syntaxe du langage des prédicats inclut les opérateurs booléens (et, ou, complément...) ; ainsi que la possibilité de construire de nouveaux prédicats à partir des prédicats existants en utilisant les modificateurs `BeginOf_` (et `EndOf_`), qui permettent

[1] La notion de capture d'un sous-mot par une expression est définie comme le plus long sous-mot appartenant au langage dénoté par cette expression.

de tester si le segment considéré est au début (ou à la fin) d'un segment vérifiant le prédicat modifié. La section 2.3 illustre ces différentes possibilités.

Les prédicats servent donc à partitionner le flux et/ou à interpréter des segments du flux. Comme l'ensemble des prédicats est extensible, les prédicats sont directement utilisables pour propager des informations à l'intérieur du système de règles. Le processus de traduction y gagne en souplesse et en clarté, puisqu'il modélise explicitement le déroulement simultané de deux processus : des analyses, qui collectent dans des prédicats de nouvelles informations et des récritures du flux de symboles.

2.3. Exemples

Quelques exemples de règles sont présentés ci-dessous. Les '--' introduisent des lignes de commentaires. Le premier exemple illustre deux utilisations simples de règles à contexte rationnel :

```
-- un 'a' est transformé en un /a/ dans tout contexte.
[a] -> a;
-- un 'a' précédé de plusieurs 'r' et suivi
-- d'un 'i' ou d'un 'î' est traduit en 'E'.
r* [a] (i|î) -> E;
```

Les prédicats interviennent naturellement au niveau de l'environnement des règles :

```
-- Un s final, dans un mot pluriel, s'efface et implique
-- un contexte de liaison en 'z'
[s] ->  |- EndOf_Word /\ Plural => ZLink;
-- Règle de voisement.
[@c1] @c2 |- NotVoiced(c1) /\ Voiced(c2) => ToBeVoiced;
```

La première des deux règles illustre le processus de passage d'information entre règles : lorsque le s final est traité, on ne sait pas encore si la liaison sera réalisée, décision qui dépend en particulier de la traduction (à venir) du début du mot suivant. On se contente donc d'indiquer, par un prédicat `Zlink`, la création d'un contexte de liaison en $/z/$, qui sera ultérieurement validé ou invalidé. La seconde règle illustre l'utilisation du nommage : si l'on observe deux segments $c1$ et $c2$ qui sont respectivement non-voisés et voisés, alors la portion du flux correspondant à $c1$ est marquée comme constituant un contexte de voisement. De nouveau, la récriture effective est defférée à une règle ultérieure.

Ces exemples illustrent également quelques facilités syntaxiques du langage. Ainsi, lorsqu'un prédicat n'a pas d'argument (eg. $Zlink$ dans les exemples ci-dessus), la portion correspondante du flux d'entrée (respectivement de sortie) est prise par défaut comme argument des prédicats de la prémisse (resp. de la conclusion). Les exemples suivants illustrent le mécanisme de filtrage de prédicats :

```
-- Enumération des voyelles
predicate V = {a,â,e,é,è,ê,ë,i,î,ï,o,ô,u,û,y}
-- Enumération de tous les 'e'.
predicate E = {e,é,è,ê,ë};
-- Une voyelle, sauf un E
predicate VnotE = V \ E;
-- Entre deux voyelles, un 's' se prononce /z/.
%V [s] %V -> z;
-- A la fin d'un mot et précédé d'une voyelle,
-- 'er' se prononce /e/.
%V[e r]    -> e                    |- EndOf_Word;
-- Réfection orthographique d'un adjectif
-- finissant par 'er' (eg. 'leger')
-- Une liaison potentielle en 'r' est levée.
[%Adjective{@P{.*} e r}]  -> @P é |- => RLink;
-- Définition des syllabes ouvertes et fermées.
[%Syllable{.* %V}]         |- => OpenSyllable;
[%Syllable{.+}]            |- => ClosedSyllable;
```

Pour finir, notons que dans DIMI , il n'y a pas de distinction formelle entre règle et entrées dictionnairiques : une entrée de dictionnaire est une règle qui s'applique entre un début et une fin de mot. Quelques entrées pour le dictionnaire du français sont données ci-dessous[2] :

```
["Tréguier"] ->     t R e g y e |- => ProperNoun;
["désagrégeai"] ->  d e z a g R e Z ( e | E ) |- => Verb;
["prégnantes"] ->   p R e G a~ t @ |- => Plural /\ ZLink;
```

3. STRATÉGIES ET MODULES

3.1. Différentes granularités pour la traduction

Les règles opèrent au niveau de segments du flux. Pour un segment donné, certaines règles réussissent, d'autres au contraire échouent. Il est donc essentiel de fixer l'ordre dans lequel les règles s'appliquent. Un ordonnancement "naturel" examine les règles de la plus spécifique à la plus générale, s'assurant qu'une règle au moins réussira. Il est parfois difficile d'ordonner certaines règles, à cause d'ambiguïtés qu'on ne peut résoudre qu'*a posteriori*. DIMI introduit donc un moyen plus générique de sélectionner les règles : les stratégies, qui sont présentées à la section 3.2.

Un dernier niveau de traduction consiste à regrouper les règles et stratégies en modules (voir la Section 3.3). Par ce biais, on peut définir une transduction sous la forme d'un arrangement de boîtes noires réutilisables.

3.2. Langage de stratégies

Les règles sont des unités de traduction que l'on compose à l'aide d'opérateurs. Cette composition définit une stratégie, qui peut être appliquée à un flux pour produire un ou plusieurs flux ou bien encore un échec. Les stratégies sont connues en récriture de termes du premier ordre. Le langage de stratégies de DIMI est adapté d'un calcul de stratégies issu de [10] et fournit les opérateurs suivants :

- le choix déterministe (<+), qui applique une première stratégie et, en cas d'échec, applique la seconde. C'est le cas d'interaction entre règles le plus habituellement rencontré ;
- le choix non-déterministe (+), qui applique deux stratégies et conduit à la production de deux résultats ;
- le séquencement (;), qui applique une première stratégie sur le flux entier et, ensuite, la seconde sur le flux résultat. Cet opérateur peut être vu comme effectuant un "pipe-line" entre deux stratégies.

Pour alléger l'écriture des stratégies, DIMI fournit enfin des opérateurs de "folding" qui permettent d'exprimer, par exemple, que toutes les règles d'un module sont composées par <+ dans l'ordre dans lequel elles apparaissent.

Le langage de stratégies permet de découpler la déclaration des règles de la manière dont elles sont utilisées, augmentant ainsi la modularité du système. Définie dynamiquement au sein d'un interpréteur, une stratégie permet d'insérer des règles dans un traitement déjà défini.

Le non-déterminisme répond au désir de maintenir plusieurs résultats potentiels et différer le moment auquel un choix de traduction doit être effectué, ou pour proposer

[2] L'utilisation des guillemets est un raccourci syntaxique pour la séquence des caractères séparés par un espace.

plusieurs variantes de transcriptions. Par exemple, les différentes prononciations proposées par un dictionnaire en fonction du contexte de liaison doivent être maintenues jusqu'à ce que la décision de liaison soit prise. L'application systématique de ce mécanisme permet d'éviter de nombreux choix arbitraires. Pour contrôler la combinatoire produite par une utilisation libérale de récritures non-déterministes, les prédicats peuvent servir d'indicateurs *a priori* de la traduction voulue, fonctionnant ainsi comme des paramètres : leur présence dans la prémisse permet de filtrer les possibilités rejetées d'emblée. Ces diverses fonctionnalités du langage sont illustrées ci-dessous.

```
-- Utilisation 'classique' des règles d'un module.
-- Les règles générales du francais pour la lettre 'a'
-- sont enchaînées de manière déterministe
strategy letter_a = (<+ #G2P.letter_a.*);

-- Choix non-déterministe: diverses prononciations
-- d'un nom propre avec plusieurs jeux de règles
strategy proper = (!French.proper + !English.proper
                   + !Spanish.proper)
```

3.3. Système de modules

Un module contient des définitions de prédicats, de règles et de stratégies. Il peut importer des définitions provenant d'autres modules. Un module peut également contenir des préconditions qui factorisent une condition liée à toutes les règles du module : la condition sera vérifiée pour chacune des règles. Enfin, un module peut être compilé indépendamment des autres. On peut donc le réutiliser dans plusieurs contextes d'utilisation différents. Les modules permettent de structurer le jeu de règles et fournissent une granularité suffisante pour définir des profils de traduction.

3.4. Outils de développement

Un compilateur permet de sauvegarder les modules DIMI sous une forme opérationnelle (byte-code et automates), qui est exécutée efficacement par une machine virtuelle, dont les instructions sont des opérateurs logiques pour les prédicats, des opérateurs du calcul de stratégies et d'une instruction d'évaluation d'automates. Les automates utilisés pour compiler les expressions rationnelles étendues sont des automates finis classiques aux états desquels sont attachées des contraintes. La compilation des modules est optimisée pour tenir compte de l'interaction des règles au sein d'une stratégie, permettant par exemple de fusionner sur la base des préfixes communs un ensemble de règles ne présentant pas d'interaction, tels que, par exemple, un dictionnaire d'exception.

Un module DIMI peut être utilisé au sein d'une boucle d'interaction, d'un script shell ou sous une forme compilée. L'interaction avec d'autres outils (étiquetteur, synthétiseur) est faite en définissant des préprocesseurs et des postprocesseurs qui s'appuient sur un format permettant d'annoter des segments du flux par des prédicats. Un système de trace génère à la demande un fichier XML permettant de suivre la traduction pas à pas. Deux points de vue sont fournis : un graphe de production listant des règles *effectivement utilisées* et un graphe d'exécution montrant toutes *les règles testées*. Un traducteur XSLT permet d'en dériver un document HTML visualisable. De même, les commentaires sont extraits du code dans un format XML pour pouvoir générer automatiquement une documentation sous différents formats, par exemple pour enrichir la trace ou encore fournir une aide en ligne.

4. RÈGLES POUR LE FRANÇAIS

Nous avons utilisé DIMI pour décrire un système complet de transcripion du français, qui respecte l'organisation modulaire proposée dans [11]. Ce système comprend actuellement les modules suivants, qui sont composés par l'opérateur d'enchaînement séquentiel (;) :

- **Dict** : dictionnaire (exceptions et mots fréquents) ($\approx$ 11000 entrées).
- **Flexion** : analyse morphologique rudimentaire, qui vise à tronquer les marques de flexions "muettes' et à identifier des contextes de liaison pour les catégories grammaticale ouvertes (17 règles) ;
- **G2P** : ensemble de règles générales pour la transcription graphème-phonème (185 règles) ;
- **Phonol** : ensemble de règles phonologiques encodant par exemple les phénomènes d'ellision ou de maintien du E-muet, divers phénomènes d'assimilation (voisement...), d'harmonisation et de troncation (51 règles) ;
- **Linking** : réalisation des liaisons (7 règles) ;

5. CONCLUSION

Dans cet article, nous avons présenté DIMI , un langage et des outils conçus pour le développement de systèmes de transcription graphème-phonème. DIMI , ainsi que les règles et dictionnaires pour le français, sera distribué avec une licence libre.

RÉFÉRENCES

[1] F. Béchet. LIA_PHON : Un système complet de phonétisation de textes. *Traitement Automatique des Langues*, 42(1) :47–67, 2001.

[2] F. Béchet and F. Yvon. Les noms propres en traitement automatique de la parole. *Traitement Automatique des Langues*, 41(3) :671–707, dec 2000.

[3] N. Catach. *La phonétisation automatique du Français*. Éditions du CNRS, Paris, 1984.

[4] J. Coleman. Unification phonology : another look at 'synthesis-by-rule'. In *Proc COLING'90*, volume 3, pages 79–84, 1990.

[5] C. Consel and R. Marlet. Architecturing software using : A methodology for language development. *LNCS*, 1490 :170, 1998.

[6] Philippe Boula de Mareüil. *Étude linguistique appliqueée à la synthèse de la parole à partir du texte*. PhD thesis, Univ. Paris XI, Orsay, 1997.

[7] M. Divay. A written text processing expert system for text to phoneme conversion. In *Proc. ICSLP'90*, pages 853–856, Kobe, Japan, 1990.

[8] T. Dutoit. *An Introduction to Text-to-Speech Synthesis*. Kluwer, Dordrecht, NL, 1997.

[9] M. Mohri and R. W. Sproat. An efficient compiler for weighted rewrite rules. In *Proc. ACL*, pages 231–238, 1996.

[10] E. Visser and Z. Benaissa. A core language for rewriting. In C. Kirchner and H. Kirchner, editors, *Proc. WRLA'98*, Pont-à-Mousson, France, 1998.

[11] F. Yvon. *Prononcer par analogie : motivations, formalisations et évaluations*. PhD thesis, ENST, Paris, 1996.