Package Feature Structure
– Rapport de stage –

Auteur : Nicolas Dubois
Équipe Langue & Dialogue – LORIA (Nancy, France)

Dernière mise à jour : 1er septembre 2003

Résumé

Le package `fr.loria.led.FeatureStructure` est une API de représentation et de traitements de structure de traits. Les principaux traitements sont l'unification et la subsomption. La principale nouveauté de cette API consiste en l'extension du formalisme aux structures de traits contenant des disjonctions sur les valeurs atomique. Cette API est distribuée sous forme d'un package JAVA.
Ce package a été développé au sein de l'équipe Langue & Dialogue du LORIA (laboratoire LOrrain en Recherches Informatiques et ses Applications) de Vandœuvre-lès-Nancy, campus scientifique.

Remerciements

Je tiens à remercier particulièrement mon tuteur, Azim Roussanaly, pour m'avoir permis de faire ces deux stages au sein de son équipe. Un grand merci aussi à toutes les personnes que j'ai été amené à rencontrer durant mon stage : Pierre, Djamé, Benoît, Bertrand, Hélène, Alexandre, Yannick, ainsi que tous les autres membres de l'équipe Langue & Dialogue.

Ce document a été rédigé avec LaTeX sous GNU/Linux
à l'aide du package *avm* pour construire les structures de traits.

Chapitre 1

Les structures de traits

1.1 Définition

La définition d'une structure de traits, que nous donnons ici est inspirée de l'article de Jean Véronis [LMIV94].

Définition 1 *Une structure de traits est un ensemble* $[\, T_1, \ldots, T_n \,]$ *où chaque* T_i *est :*

- *une paire* $(\tau : \nu)$ *avec*
  - τ *: le nom du trait*
  - ν *: la valeur associé au trait*
- *une boîte de coïndexation, qui permet de faire partager la même valeur à deux traits ;*
- *un ensemble de structures de traits appelé disjonction généralisée.*

La valeur ν *d'un trait peut, quant à elle, être :*

- *atomique ;*
- *un ensemble de valeurs appelé disjonction sur valeurs ;*
- *une structure de traits.*

Exemple :

$$\begin{bmatrix} \textsc{Cat} : & \text{SN} \\ \textsc{Accord} : & \begin{bmatrix} \textsc{Nombre} : & \textsc{Singulier} \\ \textsc{Personne} : & \textsc{Troisième} \end{bmatrix} \end{bmatrix} \quad (\mathcal{D}_{SN3sg})$$

1.2 Les différents noms donnés à une structure de traits

Les structures de traits sont un outil conceptuel utilisé dans plusieurs formalismes linguistiques basés sur l'Unification. Ces dernières peuvent prendre plusieurs noms selon les formalismes :

- **structures-f** en LFG : *Lexical Functional Grammar* (Grammaire lexicale fonctionnelle) ;
- **ensembles ou matrices de traits** en GPSG *Generalized Phrase Structure Grammar* (Grammaire suntagmatique généralisée) ;
- **structures fonctionnelles** en FUG *Functional Unification Grammar* (Grammaire d'unification fonctionnelle) ;
- **termes** en DCG *Definite Clause Grammars* (Grammaires à clauses définies).

D'une manière générale, nous reprendrons la terminologie proposée dans le formalisme HPSG *Head-driven Phrase Structure Grammar* (Grammaire syntagmatique guidée par la tête) [PS94] : structures de traits ou feature structure en anglais.

1.3 Les boîtes de coïndexation

Les structures de traits ont la propriété d'être réentrantes si deux traits de cette structure partagent la même valeur. L'indication de cette valeur partagée se fait à l'aide de boîte de coïndexation. Considérons la structure de traits suivante : $\begin{bmatrix} \tau_1 : & [\alpha : \quad \beta] \\ \tau_2 : & [\alpha : \quad \beta] \end{bmatrix}$ Les deux traits τ_1 et τ_2 partagent la même valeur : $[\alpha : \quad \beta]$. Dans ce cas, il s'agit bien de deux instantes différentes. Néanmoins, si nous voulons indiquer que ces deux traits *pointent* vers la même valeur, voici la notation à utiliser : $\begin{bmatrix} \tau_1 : & \boxed{1}\,[\alpha : \quad \beta] \\ \tau_2 : & \boxed{1} \end{bmatrix}$

1.4 Graphes et structures de traits

Le logiciel PATR II développé par Stuart Shieber (1986) [Shi86] nomme les structures de traits comme étant des *dags*[1]. Cela nous permet d'introduire la manière dont les structures de traits sont implémentées dans ce package. Les arcs de ces graphes représentent les noms de traits ; les valeurs terminales du graphes sont des ensembles de valeurs atomiques. La Fig. 1.1 montre la représentation graphique de la structure de traits $\mathcal{D}$.

$$\begin{bmatrix} \textsc{Cat} : & \text{SN} & \\ \textsc{Accord} : & \boxed{1} & \begin{bmatrix} \textsc{Nombre} : & \textsc{Singulier} \\ \textsc{Personne} : & \textsc{Troisième} \end{bmatrix} \\ \textsc{Sujet} : & \boxed{1} & \end{bmatrix} \quad (\mathcal{D})$$

[1] dags : abréviation anglaise pour directed acyclic graph (graphe acyclique orienté)

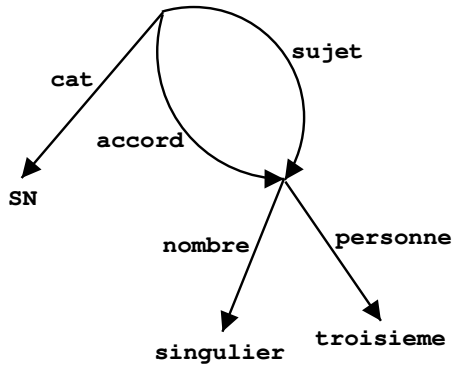


FIG. 1.1 – Représentation en graphe de $\mathcal{D}$

1.5 Les différentes fonctions des structures de traits

Les structures de traits ont essentiellement deux fonctionnalités intéressantes. Nous allons voir plus en détails chacune d'elles.

1.5.1 Fonction informative

La fonction informative, que l'on pourrait aussi appeler *fonction sémantique*, permet de dénombrer des informations sur un syntagme sous la forme de différents traits. Nous pouvons associer par exemple le syntagme "Benoît" à la structure de traits $\mathcal{D}_{SN3sg}$. Il est ainsi possible d'affirmer que "Benoît" est un syntagme de la catégorie SN (Syntagme Nominal) et qu'il se conjugue à la troisième personne du singulier.

$$\textsc{Benoît} \rightarrow \begin{bmatrix} \textsc{Cat} : & \text{SN} \\ \textsc{Accord} : & \begin{bmatrix} \textsc{Nombre} : & \textsc{Singulier} \\ \textsc{Personne} : & \textsc{Troisième} \end{bmatrix} \end{bmatrix} \quad (\mathcal{D}_{SN3sg})$$

L'extration d'information d'une structure de traits se fait à l'aide de deux fonctions :

- La fonction $D(f)$ qui renvoie la valeur associée au trait f. Par exemple, $\mathcal{D}_{SN3sg}(\textsc{Cat})$ =SN.
- La fonction dom(D) renvoie quant à elle le domaine ou l'ensemble de définition d'une structure de traits D. Par exemple, dom($\mathcal{D}_{SN3sg}$) = CAT , ACCORD .

1.5.2 Fonction grammaticale

La fonction grammaticale permet de vérifier la concordance syntaxique d'un syntagme. Par exemple, l'accord en genre et en nombre d'un nom et son déterminant se traduit par la structure de traits suivante :

$$\begin{bmatrix} \textsc{Déterminant} : & \begin{bmatrix} \textsc{Agr} : & \boxed{1} \end{bmatrix} \\ \textsc{Nom} : & \begin{bmatrix} \textsc{Agr} : & \boxed{1} \begin{bmatrix} \textsc{Nombre} : & \textsc{Singulier} \\ \textsc{Genre} : & \textsc{Masculin} \end{bmatrix} \end{bmatrix} \end{bmatrix} \quad (\mathcal{D}_{\textsc{Accord}})$$

1.6 Opérations sur les structures de traits

Les deux opérations utilisées sur les structures de traits sont la subsomption et l'unification.

1.6.1 La subsomption

Définition 2 *Une structure de traits $\mathcal{D}$ subsume une structure de traits $\mathcal{D}'$ si et seulement si $\mathcal{D}$ est plus générale (d'un point de vue grammmatical) ou contient moins d'informations (d'un point de vue sémantique) que $\mathcal{D}'$*

Notation : $\mathcal{D} \sqsubseteq \mathcal{D}'$
Exemple :

$$\begin{bmatrix} \textsc{cat} : & \text{SN} \\ \textsc{accord} : & \begin{bmatrix} \textsc{nombre} : & \textsc{singulier} \end{bmatrix} \end{bmatrix} \quad \sqsubseteq \quad \begin{bmatrix} \textsc{cat} : & \text{SN} \\ \textsc{accord} : & \begin{bmatrix} \textsc{nombre} : & \textsc{singulier} \\ \textsc{personne} : & \textsc{troisième} \end{bmatrix} \end{bmatrix}$$

1.6.2 L'unification

Définition 3 *L'unification des structures de traits $\mathcal{D}'$ et $\mathcal{D}''$ est une structure de traits $\mathcal{D}$ plus générale telle que $(\mathcal{D}' \sqsubseteq \mathcal{D}) \wedge (\mathcal{D}'' \sqsubseteq \mathcal{D})$*

Notation : $\mathcal{D} = \mathcal{D}' \sqcup \mathcal{D}''$
D'un point de vue sémantique, l'unification ajoute de l'information :

$$\begin{bmatrix} \textsc{cat} : \text{SN} \end{bmatrix} \sqcup \begin{bmatrix} \textsc{accord} : \begin{bmatrix} \textsc{nombre} : \textsc{singulier} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \textsc{cat} : & \text{SN} \\ \textsc{accord} : & \begin{bmatrix} \textsc{nombre} : \textsc{singulier} \end{bmatrix} \end{bmatrix}$$

Propriété 1 *La valeur terminale d'un trait τ commun à deux structures de traits que l'on souhaite unifier est l'intersection des deux ensembles de valeurs pour le trait τ. Si l'intersection des deux ensembles est vide, l'unification échoue.*

L'exemple suivant illustre cette propriété en mettant en plus en jeu des boîtes de coïndexation :

$$\begin{bmatrix} \textsc{Accord} : \boxed{1} \begin{bmatrix} \textsc{Nombre} : & \textsc{Singulier} \\ \textsc{Personne} : & \{\textsc{Première}, \textsc{Troisième}\} \end{bmatrix} \\ \textsc{Sujet} : \quad \begin{bmatrix} \textsc{Accord} : & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$\sqcup \quad \begin{bmatrix} \textsc{Sujet} : \begin{bmatrix} \textsc{Accord} : \begin{bmatrix} \textsc{Personne} : \textsc{Troisième} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \quad \begin{bmatrix} \textsc{Accord} : \boxed{1} \begin{bmatrix} \textsc{Nombre} : & \textsc{Singulier} \\ \textsc{Personne} : & \textsc{Troisième} \end{bmatrix} \\ \textsc{Sujet} : \quad \begin{bmatrix} \textsc{Accord} : & \boxed{1} \end{bmatrix} \end{bmatrix}$$

L'unification joue un rôle important dans l'encodage de l'information linguistique. En effet, d'un point de vue syntaxique, l'unification permet de vérifier la compatibilité des informations entre deux ou plusieurs structures de traits.

$\begin{bmatrix} \textsc{cat} : & \text{SN} \\ \textsc{accord} : & \begin{bmatrix} \textsc{nombre} : \textsc{singulier} \end{bmatrix} \end{bmatrix}$ et $\begin{bmatrix} \textsc{cat} : & \text{SN} \\ \textsc{accord} : & \begin{bmatrix} \textsc{nombre} : \textsc{pluriel} \end{bmatrix} \end{bmatrix}$ ne sont pas compatibles.

Propriété 2 *Si $\mathcal{D}' \sqsubseteq \mathcal{D}$ alors $\mathcal{D}' \sqcup \mathcal{D} = \mathcal{D} \sqcup \mathcal{D}' = \mathcal{D}$. On dit que l'unification est idempotente.*

Exemple :

$$\begin{bmatrix} \textsc{cat} : \text{SN} \end{bmatrix} \sqcup \begin{bmatrix} \textsc{cat} : & \text{SN} \\ \textsc{accord} : & \begin{bmatrix} \textsc{nombre} : \textsc{singulier} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \textsc{cat} : & \text{SN} \\ \textsc{accord} : & \begin{bmatrix} \textsc{nombre} : \textsc{singulier} \end{bmatrix} \end{bmatrix}$$

1.7 Structures de traits particulières

1.7.1 Les variables

Définition 4 *Une variable est une structure de traits dont le domaine est vide.*

Notation : $\begin{bmatrix} \quad \end{bmatrix}$ $(\mathcal{D}_{Var})$

Propriété 3 *Les variables subsument n'importe quelle structure de traits, étant donné qu'elles consistuent le cas trivial d'absence totale d'information.*
Soit $\mathcal{D}$ une structure de traits quelconque, $\mathcal{D}_{var} \sqsubseteq \mathcal{D}$ est toujours vrai.

Propriété 4 *Les variables sont l'élément d'identité de l'unification.*
Soit $\mathcal{D}$ une structure de traits quelconque, $\mathcal{D}_{var} \sqcup \mathcal{D} = \mathcal{D}$

1.7.2 Les listes

Le parser contenu dans ce package gère la notation en liste d'une structure de trait. En effet, si on considère la matrice suivante :

$$\begin{bmatrix} \textsc{Tête} : & \left\{\textsc{b,c}\right\} \\ \textsc{Reste} : & \begin{bmatrix} \textsc{Tête} : & \left[\textsc{d} : \quad \textsc{e}\right] \\ \textsc{Reste} : & \phi \end{bmatrix} \end{bmatrix} \quad (\mathcal{D}_{\text{list}})$$

La notation liste permet d'écrire $\mathcal{D}_{\text{list}}$ sous la forme : < {b ,c } , [d : e] > Ces deux structures de traits sont complètement équivalentes ; néanmoins, une liste sera représentée en interne par le package en structure de traits traditionnelles.

Chapitre 2

Package Feature Structure

2.1 Présentation

Le package `fr.loria.led.FeatureStructure` est une API de représentation et de traitements de structure de traits. Les principaux traitements sont l'unification et la subsomption. La principale nouveauté de cette API consiste en l'extension du formalisme aux structures de traits contenant des disjonctions sur les valeurs atomiques. Cette API est distribuée sous forme d'un package JAVA.

Le package FeatureStructure, développé par l'équipe Langue & Dialogue est un ensemble de classes permettant la gestion de structures de traits (réentrantes, non-typées et comportant des disjonctions sur valeurs), ainsi que de deux analyseurs syntaxiques *TermesParser* et *XmlParser*.

2.2 Utilisation

Le package est disponible sous la forme d'un fichier Jar. Il suffit de créer un programme en incluant le fichier**fs-az1.1.jar** dans le classpath de votre commande.

`[user@localhost]$ java -classpath <chemin>/fs-az1.1.jar <ma classe principale>`.

Les structures de traits sont utilisées dans plusieurs autres packages comme **segment** et **TagML2** ce qui explique qu'il soit distribué sous forme d'API.

L'ensemble des packages Java développés par l'Équipe Langue & Dialogue sont gérées par l'utilitaire **Jakarta ant**. Pour l'utiliser, il suffit de définir les 2 variables d'environnement *ANT_HOME* et *JAVA_HOME*. La syntaxe d'utilisation est :

`[user@localhost]$ ant <target>`.

L'utilitaire **ant** prend le fichier *build.xml* présent dans le répertoire courant. Pour connaître les différentes cibles (targets) disponible pour le package, taper :

`[user@localhost]$ ant usage`.

Pour voir une démonstration du package, mettant en œuvre les deux parsers (*TermesParser* et *XmlParser*), vous pouvez exécuter les classes contenues dans le sous-package `test` .

Classe	Cible	Résultat de l'exécution
Termes2FsXML	`t2fs`	Transforme l'ensemble des fichiers du répertoire `sample/termes` en fichiers XML
FsXML2Termes	`fs2t`	Transforme l'ensemble des fichiers du répertoire `sample/fsml` en fichier fs
Termes2FsLibXML	`t2fslib`	Transforme l'ensemble des fichiers du répertoire `sample/termes` en un fichier xml contenant un élément *fsLib*
FsLibXML2Termes	`fslib2t`	Transforme le fichier fslib.xml du répertoire `sample/fsml` en fichiers fs dans le répertoire `sample/fsml2termes`

FIG. 2.1 – Démonstration des parsers du package

2.3 Résumé des méthodes principales

2.3.1 Constructeur

Le constructeur de FS s'appelle simplement en faisant

```
FeatureStructure fs = new FeatureStructure();
```

D'autres constructeurs sont disponibles avec différents paramètres comme un objet **Symbole** ou encore un **SetOfSymbol**

2.3.2 Méthodes d'opérations sur les structures de traits

Les opérations d'unification et de subsomption sont disponibles dans le package. La méthode **unifier** s'applique sur un FeatureStructure[1] avec un FeatureStructure en paramètre et renvoie un FeatureStructure qui le résultat de l'unification de l'objet appliqué avec l'objet passé en paramètre. Exemple :

```
fs = fs1.unifier( fs2 );
```

La méthode **subsume** s'applique sur un FeatureStructure avec un FeatureStructure en paramètre et renvoie un booléen à vrai si l'objet auquel elle s'applique subsume l'objet passé en paramètre. Exemple :

```
if( fs1.subsume( fs2 ) ){
      /* bloc d'instructions */
}
else {
```

[1]Dans les opérations décrites, nous supposerons que les handles pris en exemple sont tous de type FeatureStructure.

```
        /* bloc d'instructions */
}
```

Les méthodes **print** et **print2XML** permettent d'imprimer à l'écran un objet FeatureStructure respectivement avec les syntaxes de TermesParser et de XmlParser. Enfin, la méthode **toXML** renvoie un objet FsXML.

La figure suivante (Fig. **??**) représente les classes principales en utilisant le langage de modélisation UML[2].

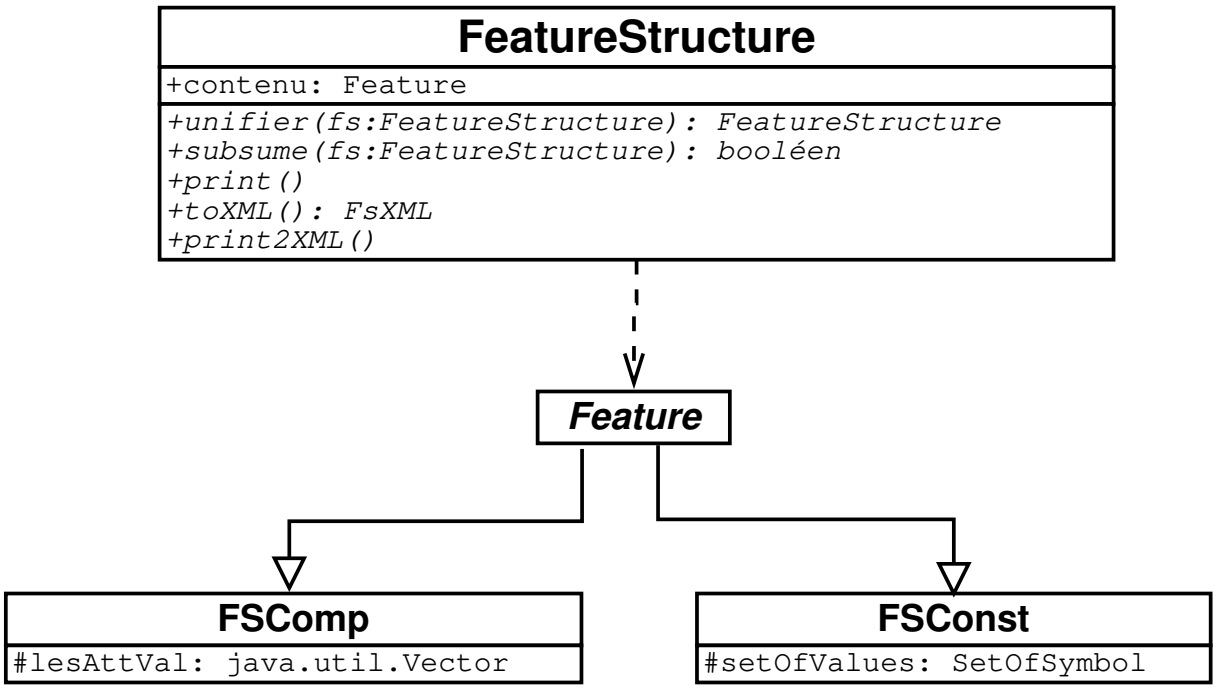


Fig. 2.2 – Schéma des différentes classes de ce package [UML]

[2]UML : Unified Modeling Language - Langage de Modélisation Objet Unifié.

Chapitre 3

TermesParser

TermesParser est un analyseur syntaxique compris dans l'API dans le sous-package `fr.loria.led.FeatureStructure.io.TermesParser` et a été développé à l'aide de l'utilitaire JavaCC[1]. La description des unités lexicales et des règles de production se fait dans un fichier généralement suffixé de l'extension *.jj*, avec une syntaxe proche du langage Java. Vous pourrez consulter ce fichier en annexe .

3.1 Définition des unités lexicales reconnues

Mnémonique	Valeurs
id	`(["a"-"Z","_","-","0"-"9", "+", "-", "\"","à"-"ÿ"])+`
etiquette	`"@" (["0"-"9"])+`
affectation	`"="`
begin_fs	`"["`
separator_fs	`";"`
end_fs	`"]"`
begin_set	`"{"`
separator_set2	`","`
end_set	`"}"`
begin_list	`"<"`
separator_list_set1	`","`
end_list	`">"`
begin_unify	`"#UNIFIER"`
separator_term	`"&"`
end_unify	`"UNIFIER#"`

FIG. 3.1 – Unités lexicales reconnues

[1] Java Compiler Compiler

3.2 Règles de réécriture

Dans cette section, nous présentons la grammaire définie dans le fichier *TermesParser.jj*. Nous prendrons les conventions suivantes :

- les non-terminaux sont en gras et en majuscules ;
- les terminaux (tokens) sont entièrement en minuscules ;
- le symbole $\longrightarrow$ se lit *se dérive en* ;

Axiome : **TERM**

TERM	$\longrightarrow$	FS_COMP | LIST | UNIFICATION
FS_COMP	$\longrightarrow$	begin_fs FS_CONTENT end_fs
FS_CONTENT	$\longrightarrow$	FS_ELEMENT separator_fs FS_CONTENT | FS_ELEMENT | VARIABLE
FS_ELEMENT	$\longrightarrow$	id affectation FEATURE_VALUE
VARIABLE	$\longrightarrow$	Λ
FEATURE_VALUE	$\longrightarrow$	TERM | FS_CONST | SHARED_VALUE
FS_CONST	$\longrightarrow$	SET1 | SET2
SET1	$\longrightarrow$	begin_set SET1_CONTENT end_set
SET2	$\longrightarrow$	SET2_CONTENT
SET1_CONTENT	$\longrightarrow$	SET_ELEMENT separator_list_set1 SET1_CONTENT | SET_ELEMENT
SET2_CONTENT	$\longrightarrow$	SET_ELEMENT separator_set2 SET2_CONTENT | SET_ELEMENT
SET_ELEMENT	$\longrightarrow$	id

SHARED_VALUE ⟶ etiquette TERM
| etiquette FS_CONST
| etiquette

LIST ⟶ begin_list LIST_CONTENT end_list

LIST_CONTENT ⟶ LIST_ELEMENT separator_list_set1 LIST_CONTENT
| LIST_ELEMENT

LIST_ELEMENT ⟶ FEATURE_VALUE

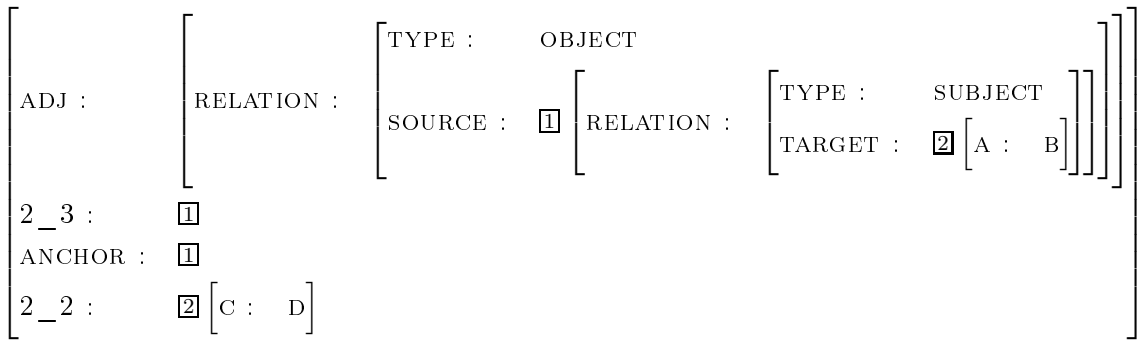
UNIFICATION ⟶ begin_unify TERM (separator_term TERM)* end_unify

La syntaxe gérée par TermesParser est assez simple et permet d'écrire rapidement des structures de traits.

3.3 Exemple de fichier

```
[ adj = [ relation = [ type = object ;
  source = @1[ relation = [ type = subject ;
  target = @2[ a = b ] ] ] ] ];
  2_3 = @1 ;
  anchor = @1 ;
  2_2 = @2[ c = d ]  ]
```

qui correspond à la structure de traits :



Chapitre 4

XmlParser

Ce chapitre est consacré à XmlParser qui est un analyseur syntaxique de structures de traits pour le langage XML. L'analyseur syntaxique en question a été créé à l'aide de l'utilitaire JAXB[1] qui est disponible dans le Java Web Services Developer Pack (JWSDP) développé par Sun Microsystems.

4.1 Définition de la grammaire du parser

La définition d'une grammaire pour le langage XML peut se faire à l'aide de 2 formalismes : les *Définitions de Types de Documents* (DTD – Document Type Definition) et les *Schémas XML*.

JAXB génère le parser à l'aide d'une grammaire définie sous la forme d'un XML-Schema. Nous allons voir la méthode utilisée pour la création du parser.

4.1.1 Définition de Type de Document

Présentation. Un groupe de recherche de l'ISO s'occupe de définir une grammaire pour les structures de traits en XML. Nous nous sommes basés sur le dernier brouillon [Lee02] qu'ils ont proposé pour définir notre DTD. Néanmoins, la grammaire proposée par le groupe de recherche rend compte des structures de traits générales, comme nous l'avons vu dans le chapître 1 ; nous avons dû adapter cette grammaire, pour qu'elle puisse convenir à la version courante de l'API FeatureStructure.

Interprétation et arrangements de la DTD. Dans l'ensemble, la DTD a été gardée comme telle pour notre parser. Nous avons interprété, ignoré et/ou ajouté certains éléments ou attributs lors des générations sémantiques dans l'analyseur syntaxique.

Attributs globaux. Pour éviter au consortium TEI d'avoir à remettre certains attributs présents dans la plupart des éléments XML, la TEI a défini plusieurs groupements d'attributs.

[1] JAXB est l'acronyme pour Java Architecture for XML Binding. Littérallement, architecture Java pour la lecture de document XML.

Ainsi, en regardant la DTD proposé dans le draft [Lee02, Annex C], on pourra remarquer la présence dans chacun des éléments XML de l'instruction : **% a.global**[2]. Pour éviter toute référence externe, nous avons remplacé toutes ces instructions *% a.global* de la DTD par le seul attribut nécessaire :

```
id  ID #IMPLIED
```

Les éléments. Dans l'API FeatureStructure, tous les éléments terminaux sont des symboles. Ainsi, tous les éléments terminaux de la DTD *(plus, minus, sym, nbr, msr, rate, str)* ont été transtypé en objet Symbole ()*sym*. Plusieurs éléments XML *(null, any, none, dft et uncertain)* correspondent à différents types de structures de traits particuliers ; nous avons choisi de les interpréter comme étant des variables (Cf. §1.7.1, page 7). Enfin, l'analyse de l'élément *fAlt*, qui rend compte des disjonctions généralisées, n'étant pas gérée par le package FeatureStructure déclenche une Exception de type *UnsupportedAlternativeFeaturesException*. Le tableau (Fig. 4.1) récapitule les interprétations des éléments XML :

Éléments	**Interprétation**
plus, minus, sym, nbr, msr, rate et str	Symbole
null, any, none, dft et uncertain	Variable
fAlt	

Fig. 4.1 – Interprétation de certains éléments.

Les attributs. Bon nombre d'attributs non gérés dans l'API ont été ignorés. Certains attributs sont communs à plusieurs éléments XML. Par exemple, les types ne sont pas gérés dans l'API actuelle. Ceux-ci sont donc simplement ignorés. De même pour l'attribut *rel* qui indique la relation entre les éléments fils de l'élément courant. Enfin, l'attribut *mutExcl* indique si les éléments d'une alternative (fAlt ou vAlt) s'excluent mutuellement ; les valeurs terminales d'un trait étant un ensemble, tous les éléments ne s'excluent pas. Le tableau (Fig. 4.2) récapitule ces attributs en question.

Attributs	**Éléments concernés**			**Interprétation**
type	fs	fLib, fsLib et fvLib	nbr, msr et rate	ignoré
rel	fs	f	sym, nbr, msr, rate et str	ignoré
mutExcl	fAlt et vAlt			toujours compris comme 'N'

Fig. 4.2 – Interprétation de certains attributs.

[2] *% a.global* signifie *Global Attributes*. Vous pouvez consulter le site internet officiel du consortium TEI pour plus informations à l'adresse : `http://www.tei-c.org/P4X/ST.html` au chapitre 3.5 Global Attributes. D'autres définitions globales comme celle-ci sont utilisées par la TEI. Exemple : % a.linking qui regroupe plusieurs attributs pour gérer les liens intra et inter-documents XML.

L'attribut *label*. Pour rendre compte des valeurs partagées[3] (Cf. §1.3, page 4), nous avons eu besoin d'ajouter un attribut qui est l'attribut **label**. Une discussion au sujet de cet attribut est prévue plus tard dans le document (Cf. §4.4.1, page 24). Voici la déclaration de l'attribut :

```
label CDATA #IMPLIED
```

L'attribut *org*. L'attribut *org* de l'élément *f* est déclaré de la manière suivante :

```
org (single | set | bag | list) #IMPLIED
```

Les valeurs *set*, *bag* et *list* sont différents types d'ensemble. Le tableau (FIG. 4.3) résume les propriétés de ces différents ensembles. On peut noter qu'une exception est générée si plusieurs

Type d'ensemble	**Ordonné**	**Répétition**
set	Oui	Non
bag	Non	Oui
list	Oui	Oui

FIG. 4.3 – Propriétés des différents ensembles *org*.

éléments *fs* sont les fils d'un élément *f*, qui a son attribut *org* affecté à set, bag ou list. Par contre, nous avons arrêté que si *org* a la valeur *single*, le parser va générer un seul objet FeatureStructure qui sera l'unification des éléments *fs*. Ceci répond notamment aux besoins de l'ancienne DTD qui possédait un élément *FsUnif*, qui permettait de renvoyer l'unification de plusieurs éléments *fs*.

4.1.2 Schéma XML.

Une fois la DTD finalisée, nous pouvons maintenant passer au schéma XML, qui est le format de description de grammaire utilisé par JAXB. Le Web est riche en documentation sur les schéma XML. Donc nous n'entrerons pas dans une description détaillée de ce formalisme. Néanmoins, nous pouvons justifier l'utilisation des schémas XML au détriment des DTDs à l'aide de ces quelques arguments :

1. les schémas XML se définissent en langage XML, et non pas dans un langage spécifique comme les DTDs ;
2. les schémas XML offrent un typage beaucoup plus riche ;
3. les schémas XML permettent de définir des contraintes.

Nous allons maintenant montrer la procédure utilisée pour passer de la DTD au schéma XML.

[3] Les valeurs partagées (shared values en anglais) se nomment aussi *boîte de coïndexation* en notation AVM (Attribute Value Matrix).

Passage de la DTD au schéma XML. La traduction de la DTD en schéma XML peut s'avérer une tâche assez laborieuse, surtout quand la grammaire est conséquente. Nous avons donc eu recours à un logiciel faisant cette traduction de manière automatique. Plusieurs logiciels de ce type sont disponibles (payant ou open source). Nous avons opté pour le logiciel XmlSpy d'Altova[4]. Rapidement, il suffit d'éditer la DTD voulue et de choisir dans le menu **DTD/Schema**, l'item **Convert DTD/Schema...** en spécifiant les options nécessaires (FIG. 4.4). Sauvegarder enfin

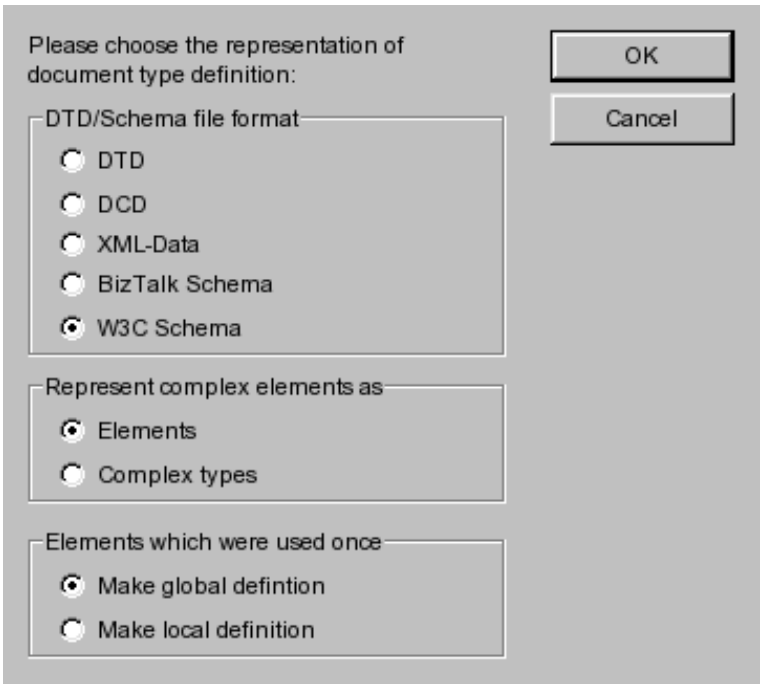


FIG. 4.4 – Options à spécifier pour la conversion.

le fichier avec l'extension généralement utilisée pour les schémas XML à savoir : **.xsd**.

Petite amélioration à effectuer manuellement. Le code généré par XmlSpy comporte quelques petites imperfections, qu'il peut être utile de corriger manuellement, en éditant le schéma dans un éditeur de texte quelconque. Nous allons corriger les redondances au niveau des séquences des éléments *fAlt* et *vAlt*. Nous prendrons l'élément *fAlt* en exemple. D'après la DTD, *fAlt* se ré-écrit en *f*, *fs* ou *fAlt*. Voici l'extrait de la DTD correspondant :

```
<!ELEMENT fAlt ((f | fs | fAlt), (f | fs | fAlt)+)>
```

On peut traduire cela, en disant qu'un *fAlt* contient au moins 2 éléments *f*, *fs* ou *fAlt*. Le code généré pour le schéma XML est le suivant :

[4]Une version d'essai peut être téléchagée sur le site d'Altova (Cf. §C, page 57)

```
<xsd:sequence>
        <xsd:choice>
                <xsd:element ref="f"/>
                <xsd:element ref="fs"/>
                <xsd:element ref="fAlt"/>
        </xsd:choice>
        <xsd:choice maxOccurs="unbounded">
                <xsd:element ref="f"/>
                <xsd:element ref="fs"/>
                <xsd:element ref="fAlt"/>
        </xsd:choice>
</xsd:sequence>
```

La petite amélioration que nous pouvons apporter, est de spécifier une seule fois l'élément *xsd :choice* en ajoutant l'attribut *minOccur* de la manière suivante :

```
<xsd:sequence>
        <xsd:choice minOccur="2" maxOccurs="unbounded">
                <xsd:element ref="f"/>
                <xsd:element ref="fs"/>
                <xsd:element ref="fAlt"/>
        </xsd:choice>
</xsd:sequence>
```

On peut effectuer la même opération pour l'élément *vAlt*.

4.2 Génération du parser

Le schéma XML finalisé, nous pouvons passer à la création du parser. L'utilitaire JAXB va générer un ensemble de classes pour analyser, valider ou encore imprimer un document XML conforme à la grammaire. Néanmoins, il est possible de personnaliser la génération pour faciliter les actions sémantiques. Cette personnalisation se fait à l'aide d'un fichier complémentaire passé en argument à JAXB. Usuellement, on lui donne le nom de **bindings.xjb**. Ce fichier est consultable à l'annexe A.3, page 42.

L'ensemble du parser est contenu dans le sous-package `fr.loria.led.FeatureStructure.io.XmlParser` . Les classes générées par JAXB sont inclues dans le sous-package `fr.loria.led.FeatureStructure.io.XmlParser.xjb` [5]. Pour faciliter l'utilisation du parser, nous avons créé 2 objets :

XmlParser qui permet d'analyser des fichiers XML, ou d'être utilisé par d'autres applications utilisant des structures de traits et créer des objets java FeatureStructure ;

XmlBuilder qui permet de construire des objets XML à partir d'objets java FeatureStructure.

[5] Le code source des classes contenues dans ce sous-package, comme l'ensemble des classes générées par JavaCC, n'est pas lisible. C'est pour cette raison que nous avons créé un sous-package supplémentaire.

4.2.1 XmlParser

Cet objet Java permet d'analyser des documents XML conformes à la DTD. Pour avoir une vision de l'ensemble des méthodes, la documentation générée par l'utilitaire JavaDoc est disponible dans l'archive contenant l'ensemble du package. Nous pouvons néanmoins donner quelques indications sur les principales méthodes à utiliser.

Méthode d'analyse. La méthode **parse()** permet d'analyser quasiment n'importe quel objet XML contenu dans la DTD (fs, fsLib, vAlt, ...). Néanmoins, il faut gérer manuellement tout un ensemble de paramètres contraignant (inclusion de packages, création d'instance de contexte...). Pour faciliter l'interfaçage, des méthodes statiques ont été créées.

Méthodes statiques d'analyse. Pour pouvoir parser un objet à l'aide du package, il existe plusieurs méthodes :

- **parseFs(FsXML)** qui renvoie un objet FeatureStructure en prenant un objet FsXML en argument ;
- **parseFsLib(FsLibXML)** qui valide un objet FsLibXML ;
- **parseFsFromFile(String)** qui analyse le fichier passé en paramètre (sous forme de chaîne de caractères) et qui renvoie un objet FeatureStructure ;
- **parseFsLibFromFile(String)** qui analyse le fichier passé en paramètre (sous forme de chaîne de caractères) et qui renvoie un objet FsLibXML.

Exemple : pour appeler une méthode de ce type, il suffit d'écrire :

```
try {
        FeatureStructure myFs = XmlParser.parseFsFromFile( myFile.xml );
} catch( UnsupportedAlternativeFeaturesException uafe ) {
        uafe.printStackTrace();
} catch( XmlParserException xpe ) {
        xpe.printStackTrace();
} catch( Exception e ) {
        e.printStackTrace();
}
```

Pour des exemples plus lisibles, vous pouvez consulter les sources des classes FsXML2Termes.java et FsLibXML2Termes.java, contenu dans le package `test` à la racine du répertoire `src` .

4.2.2 XmlBuilder

La classe XmlBuilder propose un ensemble de méthodes pour engendrer du code XML à partir d'objets Java **FeatureStructure**. Pour une facilité d'utilisation, encore une fois, la plupart d'entre elles sont déclarées en *static*. Pour plus d'information, on pourra consulter la documentation du code source générée dans le répertoire : `doc/html/` . Etant donné que le but est de générer du code XML à partir d'un objet FeatureStructure, des méthodes ont été implémentées

dans la classe FeatureStructure en faisant référance à l'objet XmlBuilder. Méthodes statiques les utilisées :

print2XML(FsXML, boolean) permet de traduire la structure de traits FsXML passée en paramètre en XML ; le booléen indent permet d'indiquer si on veut ou non indenter automatiquement le code XML.

print(FsXML) est équivalente à print2XML(FsXML, true) ; cette méthode est appelée dans la méthode FeatureStructure.print2XML().

toFsXML(FeatureStructure) renvoie un objet de type FsXML en prenant un objet FeatureStructure en argument.

Pour de plus amples détails sur les différentes méthodes et leurs utilisations, vous pouvez consulter le code source des objets Termes2FsXML.java et Termes2FsLibXML.java, contenu dans le package `test` , à la racine du répertoire `src` .

Pour scanner l'ensemble des fichiers xml[6], l'objet **XmlParserFilter** est disponible dans le sous-package `fr.loria.led.FeatureStructure.io.XmlParser` .

4.3 Utilisation du parser XML dans d'autres grammaires

Les structures de traits étant un formalisme puissant pour coder l'information ou des règles grammaticales, elles sont utilisées dans d'autres grammaires plus générales. En l'occurrence, les structures de traits sont présentes dans la grammaire de TagML2 et de segment. Nous allons voir comment il est possible d'utiliser ce parser dans ces packages.

Pour utiliser le parser, il suffit de laisser JAXB générer des classes, correspondant aux éléments compris dans la DTD des structures de traits. Ces objets ne seront pas utilisés car dans les classes type *XmlParser* et *XmlBuilder*, nous utiliserons le parser de ce package pour les productions sémantiques.

Une autre méthode pour une meilleure personnalisation d'autres parsers, aurait été d'indiquer dans le fichier de bindings les méthodes à utiliser pour l'analyse et l'impression ; exemple :

```
<xsd:complexType name="fs">
  <xsd:annotation>
    <xsd:appinfo>
      <jxb:javaType name="␣?␣"
 parseMethod="fr.loria.led.FeatureStructure.io.XmlParser.XmlParser.parse"
 printMethod="fr.loria.led.FeatureStructure.io.XmlParser.XmlBuilder.print" />
    </xsd:appinfo>
  </xsd:annotation>
</xsd:complexType>
```

Néanmoins, malgré les nombreux tests que j'ai réalisé avec la grammaire du package *segment*, aucun n'a été concluant. En effet, la documentation en ligne de JAXB sur le site de Sun Microsystems ne traite que d'exemple trop triviaux pour en comprendre le fonctionnement exact et la portée des objets que nous pouvons importés. Apparemment, seul les types primitifs et les objets les plus usuels du langage Java sont disponibles.

[6] L'extension donnée arbitrairement aux fichiers xml contenant des structures de traits est ***.fsml**.

4.4 Discussion sur la DTD

Lors de la création du parser, plusieurs points ont été discutés. Nous allons voir quelle justification nous pouvons donner à l'attribut *label* qui a été ajouté à la DTD originale et qu'elles sont les améliorations que nous pourrions proposer à la DTD.

4.4.1 Justification de l'attribut *label*

Après une étude attentive sur la grammaire proposée par le groupe de recherche de la TEI, nous n'avons pu trouver de moyen clair et précis rendant compte des valeurs partagées par plusieurs traits.

Cela nous a donc conduit à proposer un moyen d'implémenter ces dernières. Nous allons voir les 3 façons que nous avons développées avant d'arrêter celle qui nous paraissait la meilleure. Pour les illustrer, nous allons prendre pour exemple l'accord en genre et en nombre, entre un déterminant et un nom dans un syntagme nominal, codé par la structure de traits suivante :

$$\begin{bmatrix} \textsc{Déterminant} : & \boxed{1} \\ \textsc{Nom} : & \boxed{1} \begin{bmatrix} \textsc{Genre} : & \textsc{Masculin} \\ \textsc{Nombre} : & \textsc{Singulier} \end{bmatrix} \end{bmatrix} \quad (\mathcal{D}_{Acc})$$

ID/IDREFS. Tout d'abord, nous avons pensé ajouter un attribut *sameAs*[7], qui serait de type *IDREFS* et qui permettrait de pointer vers un ou plusieurs objets de type *fs*, dont le(s) ID(s) serai(en)t affecté(s) à cet attribut.

```
<fs>
  <f name="Determinant" sameAs="nom001" />
  <f name="Nom">
    <fs id="nom001">
      <f name="Genre">
        <sym value="Masculin" />
      </f>
      <f name="Nombre">
        <sym value="Singulier" />
      </f>
    </fs>
  </f>
</fs>
```

Néanmoins, cette utilisation ne nous semble pas judicieuse compte-tenu de sa ressemblance avec les attributs *feats* et *fVal*. Nous avons préféré éviter cette ambiguïté en cherchant un autre moyen.

[7] Cet attribut *sameAs* est présent dans les définitions générales de la TEI, dans le regroupement % a.linking. Ce qui explique le choix du nom de cet attribut.

X-POINTER. Notre seconde approche a été d'utiliser les X-Pointers qui sont un langage permettant d'identifier des fragments XML dans des documents.

Présentation de la méthode. Il serait nécessaire de déclarer un attribut dédié à cette référence usuellement appelé *href*. Voici notre exemple déclaré avec cet attribut :

```
<fs>
  <f name="Determinant" href=".//f[@name='Nom']" />
  <f name="Nom">
    <fs>
      <f name="Genre">
        <sym value="Masculin" />
      </f>
      <f name="Nombre">
        <sym value="Singulier" />
      </f>
    </fs>
  </f>
</fs>
```

La contrainte de déclarer un ID à un objet *fs* est levée.

Néanmoins, la déclaration d'un attribut en tant que X-Pointer ne peut se faire dans une DTD. Il est donc impossible de déclarer un attribut de type X-Pointer dans la DTD définissant les structures de traits ; il faut utiliser les schémas XML et déclarer un tel attribut en tant que type *String*.

De plus, les outils actuels pour générer des parsers ne peuvent prendre en compte ce type d'attribut. Il faut donc implémenter un sous-package dédié[8] à l'analyse d'une chaîne de caractères, pour pouvoir renvoyer l'objet auquel l'adresse du nœud indiqué par l'attribut réfère.

L'opération de construction d'un fichier XML à partir d'un objet FeatureStructure va également être rendu très complexe par l'utilisation d'un tel artifice. Enfin, ce problème d'implémentation des valeurs partagées par plusieurs soulève les limites de la représentation AVM pour les structures de traits.

Les limites de la représentation AVM Examinons notre exemple de plus près :

$$\begin{bmatrix} \text{Déterminant} : & \boxed{1} \\ \text{Nom} : & \boxed{1} \begin{bmatrix} \text{Genre} : & \text{Masculin} \\ \text{Nombre} : & \text{Singulier} \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} \text{Déterminant} : & \boxed{1} \begin{bmatrix} \text{Nombre} : & \text{Singulier} \end{bmatrix} \\ \text{Nom} : & \boxed{1} \begin{bmatrix} \text{Genre} : & \text{Masculin} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{Déterminant} : & \boxed{1} \begin{bmatrix} \text{Genre} : & \text{Masculin} \\ \text{Nombre} : & \text{Singulier} \end{bmatrix} \\ \text{Nom} : & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{Déterminant} : & \boxed{1} \begin{bmatrix} \text{Genre} : & \text{Masculin} \end{bmatrix} \\ \text{Nom} : & \boxed{1} \begin{bmatrix} \text{Nombre} : & \text{Singulier} \end{bmatrix} \end{bmatrix}$$

[8] La réalisation d'un tel package peut s'avérer délicate. Toutefois, on pourra consulter le site `http://jaxen.sf.net` , page du projet *jaxen* qui est un moteur pour l'analyse de X-Path. Ceci peut être une bonne piste si l'implémentation des X-Pointer semble être nécessaire.

Ces quatres structures de traits, qui de prime abord pourraient sembler différentes, correspondent en fait à la même structure de traits : ces quatre structures de traits sont *parfaitement équivalentes*. Cette ambiguïté est cependant levée si nous représentons ces structures de traits en graphe (FIG. 4.5).

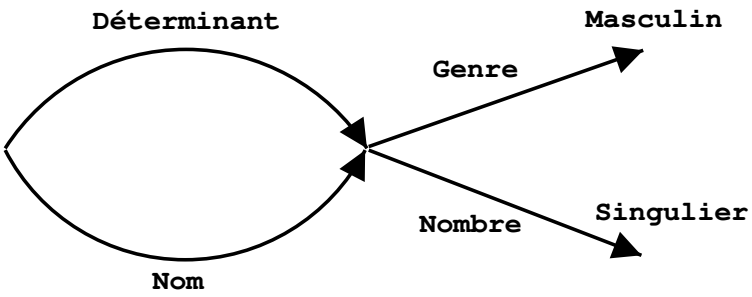


FIG. 4.5 – $(\mathcal{D}_{Acc})$ en représentation graphique

Nous illustrons ainsi les limites de la représentation AVM. Pour représenter correctement la matrice $(\mathcal{D}_{Acc})$, il faudrait, idéalement, l'écrire sous cette forme :

$$\begin{bmatrix} \text{DÉTERMINANT} : & \boxed{1} \\ \text{NOM} : & \boxed{1} \end{bmatrix} \text{ AVEC } \boxed{1} \begin{bmatrix} \text{GENRE} : & \text{MASCULIN} \\ \text{NOMBRE} : & \text{SINGULIER} \end{bmatrix}$$

Ainsi, il n'y a aucune hiérarchisation des traits à l'intérieur du terme $(\mathcal{D}_{Acc})$. Le choix des X-Pointers, ou de la solution proposée dans le sous-paragraphe ID/IDREFS ne ferait qu'accuentuer cette hiérarchisation. En effet, un élément est prit en référence et d'autres éléments, partageant sa valeur, pointeraient dessus à l'aide d'une adresse décrite dans l'attribut *href*. Cette implémentation a une, selon moi, une sémantique trop forte (FIG. 4.6). L'idée de rester au plus proche

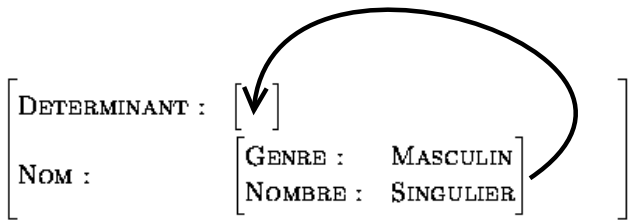


FIG. 4.6 – Représentation AVM suggérée par l'attribut *href*

de la représentation AVM habituelle...

CDATA. L'implémentation que nous avons retenue est la suivante : déclarer un attribut *label* de type CDATA qui permet de résoudre ce problème de référence. Voici un exemple d'utilisation :

```
<fs>
  <f name="Determinant" label="1" />
  <f name="Nom" label="1">
    <fs>
      <f name="Genre">
        <sym value="Masculin" />
      </f>
      <f name="Nombre">
        <sym value="Singulier" />
      </f>
    </fs>
  </f>
</fs>
```

Ainsi, nous conservons une neutralité car même si un élément conserve l'information des 2 traits *Déterminant* et *Nom*, aucun de ces 2 éléments *f* ne réfère à l'autre. Nous restons ainsi plus proche de la représentation AVM généralement admise.

Un autre argument est que cet attribut permet de rester proche de la syntaxe utiliser dans **TermesParser** (Cf. §3, page 13).

4.4.2 Propositions d'améliorations de la DTD

Ce paragraphe propose des idées pour supprimer quelques imperfections de la DTD. Étant donné que cette Définition de Type de Document gére les structures de traits de manière générale, elle rend compte des disjonctions généralisées. Toutefois, cette généralisation donne lieu a certaines redondances qui peuvent néfastes. En effet, la finalité d'une DTD est de valider un document xml conforme à la grammaire ; si celle-ci est trop ambigüe, cette validation perd tout son sens.

L'élément *vAlt*. Prenons l'exemple de l'élément *vAlt* déclaré comme suit dans la DTD :

```
<!ELEMENT vAlt
  ( (plus | minus | any | none | dft | uncertain |
     sym | nbr | msr | rate | str | vAlt | fs),
    (plus | minus | any | none | dft | uncertain |
     sym | nbr | msr | rate | str | vAlt | fs)+ )>
```

Plusieurs choses peuvent être discutées à propos de cet élément. Tout d'abord, nous avons interpréter les éléments *any*, *none*, *dft* et *uncertain*, comme étant des variables (Cf. §1.7.1, page 7). Or des variables n'ont pas de sens dans un ensemble de valeurs terminales pour un trait donné. L'attribut *vAlt* est redondant ; la présence de ce dernier induit que nous pouvons écrire ce genre de chose : $\begin{bmatrix} \textsc{Catégorie} : & \textsc{Verbe} \\ \textsc{Modes} : & \left\{ \textsc{impératif}, \left\{ \textsc{indicatif}, \textsc{conditionnel} \right\}, \textsc{subjonctif} \right\} \end{bmatrix}$

Formellement, un tel ensemble $\{imperatif, \{indicatif, conditionnel\}, subjonctif\}$ n'est pas gênant : il suffit de considérer l'union de tous les ensembles de façon récursive, obtenant ainsi :

$\{imperatif, indicatif, conditionnel, subjonctif\}$. Néanmoins, nous ne comprenons pas le sens a donné à un tel ensemble.

L'occurrence *fs* est également redondante : l'élément *fAlt* permet de créer une alternative de traits, il n'est donc pas nécessaire d'en ajouter l'option dans l'élément *vAlt*, qui devrait se limiter aux valeurs terminales d'un trait.

Enfin, la justification d'au moins 2 occurrences de valeurs terminales ne semble pas pertinente : la valeur atomique d'un trait peut très bien être considérée comme un singleton ($\nu \leftrightarrow \{\nu\}$).

L'élément *fAlt*. Comme pour l'élément *vAlt*, l'élément peut être récursif. Encore une fois, un ensemble d'ensembles ne semble pas judicieux, car dépourvu de sens.

L'élément *f*. L'élément *f* se déclare de cette manière :

```
<!ELEMENT f (null | (plus | minus | any | none
        | dft | uncertain | sym | nbr | msr | rate | str | vAlt | fs)*)>
<!ATTLIST f
        id ID #IMPLIED
        name NMTOKEN #REQUIRED
        label CDATA #IMPLIED
        org (single | set | bag | list) #IMPLIED
        rel (eq | ne | sb | ns) "eq"
        fVal IDREFS #IMPLIED
>
```

Cela implique qu'en affectant l'attribut *org* a une valeur attendue, il est possible d'écrire autant d'éléments terminaux souhaiter, même sans les inclure dans un élément *vAlt*. Ainsi, quelle différence existe-t-il entre ces deux structures conformes à la DTD :

```
<f name="personne" org="set">
  <sym value="premiere" />
  <sym value="troisieme" />
</f>

<f name="personne">
  <vAlt>
    <sym value="premiere" />
    <sym value="troisieme" />
  </vAlt>
</f>
```

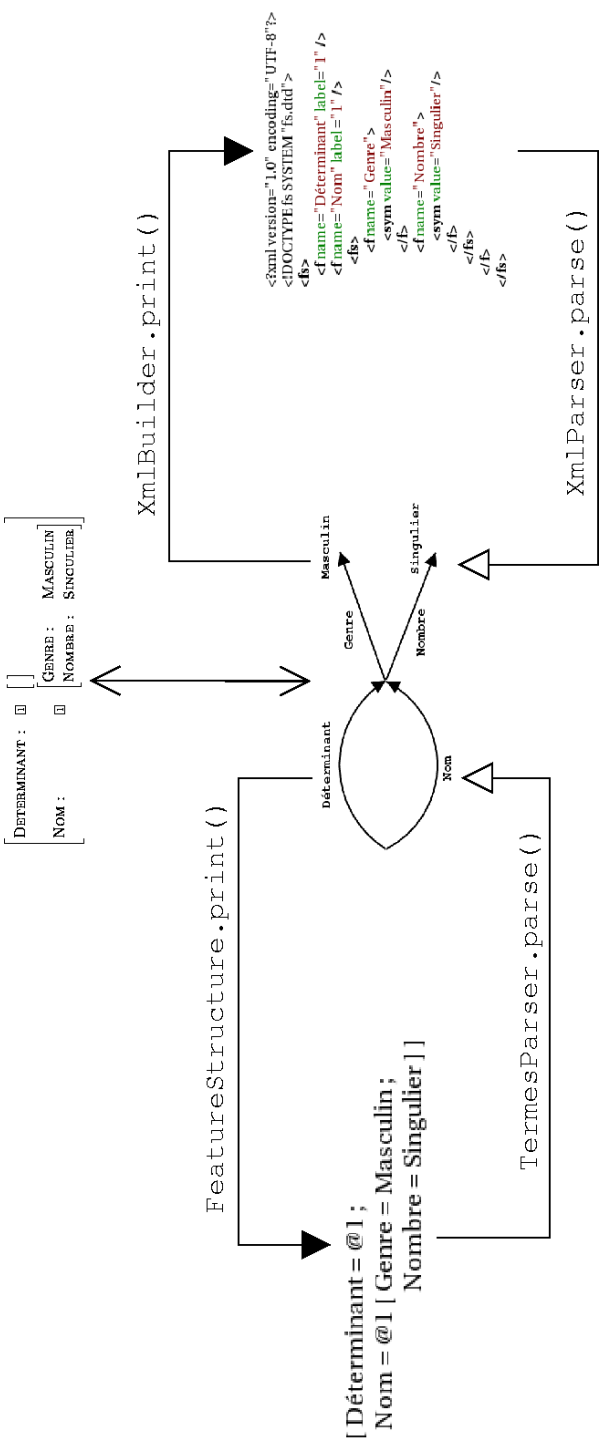


FIG. 4.7 – Schéma récapitulatif du package

Annexe A

Sources XML

A.1 Document Type Definition

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!ENTITY % FS "-//INRIA-LORIA LED//DTD FS v0.1 Draft//EN">

<!ELEMENT fs ((f | fAlt)*)>
<!ATTLIST fs
        id ID #IMPLIED
        type CDATA #IMPLIED
        feats IDREFS #IMPLIED
        rel (eq | ne | sb | ns) "sb">

<!ELEMENT f (null |
        (plus | minus | any | none | dft | uncertain | sym
        | nbr | msr | rate | str | vAlt | fs)*)>
<!ATTLIST f
        id ID #IMPLIED
        name NMTOKEN #REQUIRED
        label CDATA #IMPLIED
        org (single | set | bag | list) #IMPLIED
        rel (eq | ne | sb | ns) "eq"
        fVal IDREFS #IMPLIED>

<!ELEMENT plus EMPTY>
<!ATTLIST plus  id ID #IMPLIED>

<!ELEMENT minus EMPTY>
<!ATTLIST minus id ID #IMPLIED>

<!ELEMENT fLib ((f | fAlt)*)>
```

```
<!ATTLIST fLib type CDATA #IMPLIED>

<!ELEMENT fsLib ((fs | vAlt)*)>
<!ATTLIST fsLib type CDATA #IMPLIED>

<!ELEMENT fvLib ((sym | nbr | msr | rate | str | vAlt)*)>
<!ATTLIST fvLib type CDATA #IMPLIED>

<!ELEMENT sym EMPTY>
<!ATTLIST sym
        id ID #IMPLIED
        value CDATA #REQUIRED
        rel (eq | ne) "eq">

<!ELEMENT nbr EMPTY>
<!ATTLIST nbr
        id ID #IMPLIED
        value CDATA #REQUIRED
        valueTo CDATA #IMPLIED
        rel (eq | ne | lt | le | gt | ge) "eq"
        type (int | real) #IMPLIED>

<!ELEMENT msr EMPTY>
<!ATTLIST msr
        id ID #IMPLIED
        value CDATA #REQUIRED
        valueTo CDATA #IMPLIED
        unit CDATA #REQUIRED
        rel (eq | ne | lt | le | gt | ge) "eq"
        type (int | real) #IMPLIED>

<!ELEMENT rate EMPTY>
<!ATTLIST rate
        id ID #IMPLIED
        value CDATA #REQUIRED
        valueTo CDATA #IMPLIED
        unit CDATA #IMPLIED
        per CDATA #REQUIRED
        rel (eq | ne | gt | ge | lt | le) "eq"
        type (int | real) #IMPLIED>

<!ELEMENT str (#PCDATA)>
<!ATTLIST str
        id ID #IMPLIED
```

```
        rel (eq | ne | sb | ns | lt | le | gt | ge) "eq">

<!ELEMENT null EMPTY>
<!ATTLIST null id ID #IMPLIED>


<!ELEMENT fAlt ((f | fs | fAlt), (f | fs | fAlt)+)>
<!ATTLIST fAlt mutExcl (Y | N) #IMPLIED>

<!ELEMENT vAlt (
        (plus | minus | any | none | dft | uncertain | sym
        | nbr | msr | rate | str | vAlt | fs),
        (plus | minus | any | none | dft | uncertain | sym
        | nbr | msr | rate | str | vAlt | fs)+)>
<!ATTLIST vAlt
        id ID #IMPLIED
        mutExcl (Y | N) #IMPLIED>

<!ELEMENT any EMPTY>
<!ATTLIST any id ID #IMPLIED>

<!ELEMENT none EMPTY>
<!ATTLIST none id ID #IMPLIED>

<!ELEMENT dft EMPTY>
<!ATTLIST dft id ID #IMPLIED>

<!ELEMENT uncertain EMPTY>
<!ATTLIST uncertain id ID #IMPLIED>
```

A.2 Schéma XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xsd:element name="any">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="dft">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="f">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element ref="null"/>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="plus"/>
          <xsd:element ref="minus"/>
          <xsd:element ref="any"/>
          <xsd:element ref="none"/>
          <xsd:element ref="dft"/>
          <xsd:element ref="uncertain"/>
          <xsd:element ref="sym"/>
          <xsd:element ref="nbr"/>
          <xsd:element ref="msr"/>
          <xsd:element ref="rate"/>
          <xsd:element ref="str"/>
          <xsd:element ref="vAlt"/>
          <xsd:element ref="fs"/>
        </xsd:choice>
      </xsd:choice>
      <xsd:attribute name="id" type="xsd:ID"/>
      <xsd:attribute name="name" type="xsd:NMTOKEN" use="required"/>
      <xsd:attribute name="label" type="xsd:string"/>
      <xsd:attribute name="org">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="single"/>
            <xsd:enumeration value="set"/>
```

```
            <xsd:enumeration value="bag"/>
            <xsd:enumeration value="list"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="rel" default="eq">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="eq"/>
            <xsd:enumeration value="ne"/>
            <xsd:enumeration value="sb"/>
            <xsd:enumeration value="ns"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="fVal" type="xsd:IDREFS"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="fAlt">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="2" maxOccurs="unbounded">
          <xsd:element ref="f"/>
          <xsd:element ref="fs"/>
          <xsd:element ref="fAlt"/>
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="mutExcl">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="Y"/>
            <xsd:enumeration value="N"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="fLib">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="f"/>
        <xsd:element ref="fAlt"/>
      </xsd:choice>
      <xsd:attribute name="type" type="xsd:string"/>
```

```
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="fs">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="f"/>
        <xsd:element ref="fAlt"/>
      </xsd:choice>
      <xsd:attribute name="id" type="xsd:ID"/>
      <xsd:attribute name="type" type="xsd:string"/>
      <xsd:attribute name="feats" type="xsd:IDREFS"/>
      <xsd:attribute name="rel" default="sb">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="eq"/>
            <xsd:enumeration value="ne"/>
            <xsd:enumeration value="sb"/>
            <xsd:enumeration value="ns"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="fsLib">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="fs"/>
        <xsd:element ref="vAlt"/>
      </xsd:choice>
      <xsd:attribute name="type" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="fvLib">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="sym"/>
        <xsd:element ref="nbr"/>
        <xsd:element ref="msr"/>
        <xsd:element ref="rate"/>
        <xsd:element ref="str"/>
        <xsd:element ref="vAlt"/>
      </xsd:choice>
      <xsd:attribute name="type" type="xsd:string"/>
    </xsd:complexType>
```

```
</xsd:element>
<xsd:element name="minus">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="msr">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:ID"/>
    <xsd:attribute name="value" type="xsd:string" use="required"/>
    <xsd:attribute name="valueTo" type="xsd:string"/>
    <xsd:attribute name="unit" type="xsd:string" use="required"/>
    <xsd:attribute name="rel" default="eq">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="eq"/>
          <xsd:enumeration value="ne"/>
          <xsd:enumeration value="lt"/>
          <xsd:enumeration value="le"/>
          <xsd:enumeration value="gt"/>
          <xsd:enumeration value="ge"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="type">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="int"/>
          <xsd:enumeration value="real"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
<xsd:element name="nbr">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:ID"/>
    <xsd:attribute name="value" type="xsd:string" use="required"/>
    <xsd:attribute name="valueTo" type="xsd:string"/>
    <xsd:attribute name="rel" default="eq">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="eq"/>
          <xsd:enumeration value="ne"/>
```

```
            <xsd:enumeration value="lt"/>
            <xsd:enumeration value="le"/>
            <xsd:enumeration value="gt"/>
            <xsd:enumeration value="ge"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="type">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="int"/>
            <xsd:enumeration value="real"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="none">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="null">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="plus">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="rate">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
      <xsd:attribute name="value" type="xsd:string" use="required"/>
      <xsd:attribute name="valueTo" type="xsd:string"/>
      <xsd:attribute name="unit" type="xsd:string"/>
      <xsd:attribute name="per" type="xsd:string" use="required"/>
      <xsd:attribute name="rel" default="eq">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="eq"/>
            <xsd:enumeration value="ne"/>
```

```
            <xsd:enumeration value="gt"/>
            <xsd:enumeration value="ge"/>
            <xsd:enumeration value="lt"/>
            <xsd:enumeration value="le"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="type">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="int"/>
            <xsd:enumeration value="real"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="str">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:string">
          <xsd:attribute name="id" type="xsd:ID"/>
          <xsd:attribute name="rel" default="eq">
            <xsd:simpleType>
              <xsd:restriction base="xsd:NMTOKEN">
                <xsd:enumeration value="eq"/>
                <xsd:enumeration value="ne"/>
                <xsd:enumeration value="sb"/>
                <xsd:enumeration value="ns"/>
                <xsd:enumeration value="lt"/>
                <xsd:enumeration value="le"/>
                <xsd:enumeration value="gt"/>
                <xsd:enumeration value="ge"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="sym">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
      <xsd:attribute name="value" type="xsd:string" use="required"/>
```

```
      <xsd:attribute name="rel" default="eq">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="eq"/>
            <xsd:enumeration value="ne"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
      </xsd:complexType>
  </xsd:element>
  <xsd:element name="uncertain">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="vAlt">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="2" maxOccurs="unbounded">
          <xsd:element ref="plus"/>
          <xsd:element ref="minus"/>
          <xsd:element ref="any"/>
          <xsd:element ref="none"/>
          <xsd:element ref="dft"/>
          <xsd:element ref="uncertain"/>
          <xsd:element ref="sym"/>
          <xsd:element ref="nbr"/>
          <xsd:element ref="msr"/>
          <xsd:element ref="rate"/>
          <xsd:element ref="str"/>
          <xsd:element ref="vAlt"/>
          <xsd:element ref="fs"/>
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID"/>
      <xsd:attribute name="mutExcl">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="Y"/>
            <xsd:enumeration value="N"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
```

```
    </xsd:element>
</xsd:schema>
```

A.3 Fichier de bindings pour JAXB

```
<?xml version="1.0" encoding="iso-8859-1"?>
<jxb:bindings
   version="1.0"
   xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <jxb:bindings
      schemaLocation="feature-structure.xsd"
      node="/xsd:schema">
      <jxb:globalBindings
         collectionType="java.util.Vector"
         fixedAttributeAsConstantProperty="false"
         generateIsSetMethod="false"
         enableFailFastCheck="false"
         choiceContentProperty="false"
         underscoreBinding="asCharInWord"
         enableJavaNamingConventions="true"
         typesafeEnumBase="xsd:NCName"
         typesafeEnumMemberName="generateError"
         bindingStyle="elementBinding"/>
      <jxb:schemaBindings>
         <jxb:package
            name="fr.loria.led.FeatureStructure.io.XmlParser.xjb">
            <jxb:javadoc>
               <![CDATA[
                  <html>
                     <body>
                        Package for parsing xml feature structure files.
                     </body>
                  </html>
               ]]>
            </jxb:javadoc>
         </jxb:package>
         <jxb:nameXmlTransform>
            <jxb:elementName suffix="XML"/>
         </jxb:nameXmlTransform>
      </jxb:schemaBindings>
   </jxb:bindings>
</jxb:bindings>
```

Annexe B

Sources Java

B.1 TermesParser.jj

```
options {
    LOOKAHEAD = 1;
    CHOICE_AMBIGUITY_CHECK = 2;
    OTHER_AMBIGUITY_CHECK = 1;
    STATIC = false;
    DEBUG_PARSER = false;
    DEBUG_LOOKAHEAD = false;
    DEBUG_TOKEN_MANAGER = false;
    ERROR_REPORTING = true;
    JAVA_UNICODE_ESCAPE = false;
    UNICODE_INPUT = false;
    IGNORE_CASE = false;
    USER_TOKEN_MANAGER = false;
    USER_CHAR_STREAM = false;
    BUILD_PARSER = true;
    BUILD_TOKEN_MANAGER = true;
    SANITY_CHECK = true;
    FORCE_LA_CHECK = false;
}

PARSER_BEGIN(TermesParser)

package fr.loria.led.FeatureStructure.io.TermesParser;

import java.util.Vector;
import java.util.Enumeration;

import fr.loria.led.FeatureStructure.FeatureStructureConstants;
```

```
import fr.loria.led.FeatureStructure.TD_DefEtiquettes;
import fr.loria.led.FeatureStructure.Correspondances;
import fr.loria.led.FeatureStructure.SetOfSymbol;
import fr.loria.led.FeatureStructure.Symbole;
import fr.loria.led.FeatureStructure.FeatureStructure;

public class TermesParser {

    public TD_DefEtiquettes tde = new TD_DefEtiquettes();
    public Correspondances tcorr = new Correspondances();

    public FeatureStructure tryToParse(TD_DefEtiquettes table )
        throws ParseException {
        tde = table;
        return this.TERM();
    }

    public FeatureStructure tryToParse()
        throws ParseException{
        return tryToParse(new TD_DefEtiquettes());
    }

    public FeatureStructure parse(TD_DefEtiquettes table){
        try{
          return tryToParse(table);
        }catch(ParseException e){
           return null;
        }
    }

    public FeatureStructure parse(){
        try{
          return tryToParse();
        }catch(ParseException e){
           return null;
        }
    }

    public FeatureStructure analyser(){
      return parse();
    }

    public FeatureStructure analyser(TD_DefEtiquettes d){
      return parse(d);
```

```
    }

    public FeatureStructure ajoutParUnification(FeatureStructure fs1,
                FeatureStructure fs2)
        throws ParseException{
        if (! fs1.unifierAvec(fs2,tcorr))
            throw new ParseException("unification impossible
                                    (ligne : "+String.valueOf(token.beginLine)+
                ", colonne : "+String.valueOf(token.beginColumn)+").");
        return fs1;
    }
}

PARSER_END(TermesParser)

SKIP : {
                " " | "\t" | "\n" | "\r"
}

TOKEN : {
    < id: ( ["a"-"z","A"-"Z","_","-","0"-"9", "+", "-", "\"","ï¿½"-"ï¿½"] )+ >
        |
        < etiquette: "@" ( ["0"-"9"] )+ >
        |
        < affectation: "=" >
        |
        < begin_fs: "[" >
        |
        < separator_fs: ";" >
        |
        < end_fs: "]" >
        |
        < separator_set2: "|" >
        |
        < begin_set: "{" >
        |
        < end_set: "}" >
        |
        < begin_list: "<" >
        |
        < separator_list_set1: "," >
        |
        < end_list: ">" >
        |
```

```
        < begin_unifier : "#UNIFIER" >
        |
        < end_unifier : "UNIFIER#" >
        |
        < separator_term : "&" >
}

FeatureStructure TERM() : {
    FeatureStructure result = null;
}
{
    result = FS_COMP() {
        return result;
    }
    |
        result = LIST() {
            return result;
        }
        |
            result = UNIFICATION() {
                return result;
            }
}

FeatureStructure FS_COMP() : {
    FeatureStructure result = null;
}
{
    < begin_fs >
    result = FS_CONTENT()
    < end_fs > {
              return result;
    }
}

FeatureStructure FS_CONTENT()
        throws ParseException : {
    FeatureStructure fsElement = null;
    FeatureStructure result = new FeatureStructure();
}
{
    fsElement = FS_ELEMENT() {
        result = ajoutParUnification(result,fsElement);
    }
```

```
    [ < separator_fs >
        fsElement = FS_CONTENT() {
            result = ajoutParUnification(result,fsElement);
    }
    ]{
        return result;
    }
    |
      fsElement = VARIABLE()
    {
        return fsElement;
    }
}

FeatureStructure VARIABLE() : {
}
{
  {
    return new FeatureStructure();
  }
}

FeatureStructure FS_ELEMENT() : {
    Symbole nomAttribut;
    FeatureStructure featureValue = null;
    FeatureStructure result = new FeatureStructure();
}
{
    < id > {
        nomAttribut = new Symbole(token.image);
    }
    < affectation >
    featureValue = FEATURE_VALUE() {
              result.addAttValPair(nomAttribut,featureValue);
              return result;
    }
}

FeatureStructure FEATURE_VALUE() : {
    FeatureStructure result = null;
}
{
    result = TERM() {
        return result;
```

```
    }
    |
    result = FS_CONST() {
        return result;
    }
    |
    result = SHARED_VALUE() {
        return result;
    }
}

FeatureStructure FS_CONST() : {
    FeatureStructure fsConst = null;
}
{
    fsConst = SET1() {
        return fsConst;
    }
    |
    fsConst = SET2() {
        return fsConst;
    }
}

FeatureStructure SET1() : {
    SetOfSymbol set1 = null;
}
{
    < begin_set >
    set1 = SET1_CONTENT()
    < end_set > {
            return new FeatureStructure(set1);
    }
}

FeatureStructure SET2() : {
    SetOfSymbol set2 = null;
}
{
    set2 = SET2_CONTENT() {
        return new FeatureStructure(set2);
    }
}
```

```
SetOfSymbol SET1_CONTENT() : {
    Symbole setElement;
    SetOfSymbol set1 = new SetOfSymbol();
}
{
    setElement = SET_ELEMENT() {
        set1.addElement(setElement);
    }
    [ < separator_list_set1 >
    set1 = SET1_CONTENT() {
        set1.addElement(setElement);
    } ] {
        return set1;
    }
}

SetOfSymbol SET2_CONTENT() : {
    Symbole setElement;
    SetOfSymbol set2 = new SetOfSymbol();
}
{
    setElement = SET_ELEMENT() {
        set2.addElement(setElement);
    }
    [ < separator_set2 >
    set2 = SET2_CONTENT() {
        set2.addElement(setElement);
    } ] {
        return set2;
    }
}

Symbole SET_ELEMENT() : {
}
{
    < id > {
        return new Symbole(token.image);
    }
}

FeatureStructure SHARED_VALUE()
        throws ParseException : {
    String etiquetteLue;
    FeatureStructure result = null;
```

```
    FeatureStructure fs = new FeatureStructure();
}
{
    < etiquette > {
        etiquetteLue = token.image;
        if(tde.etiquetteEstDefinie(etiquetteLue)){
            result = tde.valEtiquette(etiquetteLue);
        }
        else {
            result = new FeatureStructure();
            tde.ajouterDefinition(etiquetteLue,result);
        }
    }
    [ fs = TERM()
      |
      fs = FS_CONST()
    ] {
        result = ajoutParUnification(result,fs);
        tde.ajouterDefinition(etiquetteLue,result);
        return result;
    }
}

FeatureStructure LIST() : {
    FeatureStructure result = null;
}
{
    < begin_list >
    result = LIST_CONTENT()
    < end_list > {
            return result;
    }
}

FeatureStructure LIST_CONTENT()
        throws ParseException: {
    FeatureStructure tmp = new FeatureStructure();
    FeatureStructure head = new FeatureStructure();
    FeatureStructure tail = new FeatureStructure();
    FeatureStructure endOfList = new FeatureStructure(new Symbole(END_OF_LIST))
}
{
    head = LIST_ELEMENT()
        {
```

```
            tmp.addAttValPair(HEAD,head);
            head = tmp;
        }
    [ < separator_list_set1 >
    tail = LIST_CONTENT() {
        tmp.addAttValPair(TAIL,tail);
        tail = tmp;
        return ajoutParUnification(head,tail);
    } ] {
        tail.addAttValPair(new Symbole(TAIL),endOfList);
        return ajoutParUnification(head,tail);
    }
}

FeatureStructure LIST_ELEMENT() : {
    FeatureStructure result = null;
}
{
    result = FEATURE_VALUE() {
        return result;
    }
}

FeatureStructure UNIFICATION() : {
    FeatureStructure result = null;
    FeatureStructure tmp = null;
}
{
    < begin_unifier >
          result = TERM()
          < separator_term > tmp = TERM(){
              result = ajoutParUnification(result,tmp);
          }
    ( < separator_term > tmp = TERM() {
        result = ajoutParUnification(result,tmp);
    } ) *
        < end_unifier >
        { return result; }
}
```

B.2 Termes2FsXML.java

```
package test;

// imports from package:
import fr.loria.led.FeatureStructure.FeatureStructure;
import fr.loria.led.FeatureStructure.io.TermesParser.TermesParser;
import fr.loria.led.FeatureStructure.io.TermesParser.TermesParserFilter;
import fr.loria.led.FeatureStructure.io.XmlParser.XmlBuilder;
import fr.loria.led.FeatureStructure.util.TextFile;

// imports from java api 1.4.1:
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public final class Termes2FsXML {

  public static void main( String [] argv ){

    try{
      XmlBuilder.version();
      String path = "./samples";
      File inputDirectory = new File( path+"/termes" );
      File outputDirectory = new File( path+"/termes2fsml" );
      if( inputDirectory.isDirectory() ) {
        String [] files = inputDirectory.list( new TermesParserFilter() );
        for( int i = 0; i < files.length; i++ ) {
          String inputFile = inputDirectory+"/"+files[i];
          String outputFile = outputDirectory+"/"+files[i]+"ml";
          FeatureStructure fs = new TermesParser(
              new FileInputStream( inputFile ) ).parse();
          System.out.println( "\n--- "+int2ordinal( i+1 )
                              +" feature structure ----------------" );
          System.out.println( "\nParsing input file:\n\t"+inputFile );
          System.out.println( "\nTerme parsed:" );
          new TextFile( inputFile ).print();
          System.out.println( "\nFeature structure created:" );
          fs.print();
          System.out.println( "\nXML transformation:" );
          XmlBuilder.print( XmlBuilder.toFsXML( fs ) );
          System.out.println( "Creating output file:\n\t"+outputFile );
          XmlBuilder.print2XMLIntoFile
```

```
                ( XmlBuilder.toFsXML( fs ), new File( outputFile ) );
            }
          }
          else {
            throw new IOException( "Directory "+inputDirectory+" doesn't exist." );
          }
        } catch( IOException ioe ){
          System.err.println( ioe.getMessage() );
        } catch(Exception e){
          e.printStackTrace();
        }
      }

      private static String int2ordinal( int value ) {
        if( value == 1 ) return "1st";
        if( value == 2 ) return "2nd";
        if( value == 3 ) return "3rd";
        if( value / 10 > 1 ) return (value / 10)+int2ordinal( value % 10 );
        return value+"th";
      }

}
```

B.3 FsXML2Termes.java

```
package test;

// imports from package:
import fr.loria.led.FeatureStructure.FeatureStructure;
import fr.loria.led.FeatureStructure.io.XmlParser.XmlParser;
import fr.loria.led.FeatureStructure.io.XmlParser.XmlParserFilter;
import fr.loria.led.FeatureStructure.util.TextFile;

// imports from java api 1.4.1:
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public final class FsXML2Termes {

  public static void main( String [] argv ){

    try{
      XmlParser.version();
      String path = "./samples";
      File inputDirectory = new File( path+"/fsml" );
      File outputDirectory = new File( path+"/fsml2termes" );
      if( inputDirectory.isDirectory() ) {
        String [] files = inputDirectory.list( new XmlParserFilter() );
        for( int i = 0; i < files.length; i++ ) {
          String inputFile = inputDirectory+"/"+files[i];
          String outputFile = outputDirectory+"/"
                        +files[i].replaceAll( "ml$", "" );
          FeatureStructure fs = XmlParser.parseFsFromFile( inputFile );
          fs.print();
          XmlParser.parseFsFromFile( inputFile ).print();
          System.out.println( "\n---␣"+int2ordinal( i+1 )
                              +"␣feature␣structure␣----------------" );
          System.out.println( "\nParsing␣input␣file:\n\t"+inputFile );
          System.out.println( "\nXml␣file␣parsed:" );
          new TextFile( inputFile ).print();
          System.out.println( "\nFeature␣structure␣created:" );
          fs.print();
          System.out.println( "Creating␣output␣file:\n\t"+outputFile );
          new TextFile( outputFile ).fromString( fs.toString() );
        }
```

```
        }
      } catch(Exception e){
        e.printStackTrace();
      }
    }

    private static String int2ordinal( int value ) {
      if( value == 1 ) return "1st";
      if( value == 2 ) return "2nd";
      if( value == 3 ) return "3rd";
      if( value / 10 > 1 ) return (value / 10)+int2ordinal( value % 10 );
      return value+"th";
    }

}
```

Annexe C

Ressources diverses

C.1 Ressources logicielles

Cette section récapitule les logiciels utilisés pour le développement de ce package.

JDK : Java Development Kit (version utilisée : 1.4.1)
`http://java.sun.com/j2se/1.4.1/download.html`

Le langage Java et la plupart des outils utilisés pour le développement de cette API sont propriétés de la société *Sun Microsystems*.

JWSDP : Java Web Services Developer Pack : comprenant JAXB et ant (version utilisée : 1.1)
`http://java.sun.com/webservices/download.html`

JavaCC : Java Compiler Compiler (version utilisée : 2.1)
`https://javacc.dev.java.net/`

XmlSpy : Logiciel d'Altova permettant de convertir une DTD en schéma XML (version utilisée : 5 rc4)
`http://www.altova.com/default.asp`

C.2 Ressources en ligne

Cette section indique quelques pointeurs utiles pour des informations complémentaires sur les concepts abordés dans le développement de cette API.

C.2.1 Ressources Java

JDK : Documentation de l'API Java utilisée
`http://java.sun.com/j2se/1.4.1/docs/api/`

JWSDP : Tutorial du Java Web Services Developer Pack
`http://java.sun.com/webservices/docs/1.1/tutorial/doc/`

JavaCC : Java Compiler Compiler
`https://javacc.dev.java.net/doc/docindex.html`

C.2.2 Ressources XML

Schéma XML : Documentation officielle sur les schémas XML
`http://xmlfr.org/w3c/TR/xmlschema-0/`

DTD : Documentation sur les Définitions de Type de Document
`http://xmlfr.org/index/object.title/dtd/`

W3C : Consortium se chargeant de normaliser les langages pour le Web
`http://www.w3.org`

C.2.3 Ressources sur les structures de traits

TEI : Documentation du consortium TEI
`http://www.tei-c.org/Guidelines/FS.html`

TC 37 SC 4 : Groupe de recherche de l'ISO pour la normalisation des structures de traits
`http://www.tc37sc4.org/`

Références bibliographiques

[Lee02] Kiyong Lee. Language Resource Management - Feature Structures. Draft, ISO (Organisation Internationale de Normalisation), août 2002. Référence : ISO TC 37/SC4 N0X.

[LMIV94] Jacques Le Maitre, Nancy Ide, and Jean Véronis. Modélisation et interrogation de bases de données lexicales. *Ingénierie des Systèmes d'Information*, 2(1) :57–82, 1994.

[MT90] Philip Miller and Thérèse Torris. *Formalismes syntaxiques pour le traitement automatique du langage naturel*. Hermès, 1990.

[PS94] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, 1994.

[Shi86] Stuart Shieber. *An introduction to Unification – Based approaches to Grammar*. The University of Chicago, 1986.

Table des figures

1.1 Représentation en graphe de $\mathcal{D}$ 5

2.1 Démonstration des parsers du package 10
2.2 Schéma des différentes classes de ce package [UML] 11

3.1 Unités lexicales reconnues 13

4.1 Interprétation de certains éléments. 18
4.2 Interprétation de certains attributs. 18
4.3 Propriétés des différents ensembles *org*. 19
4.4 Options à spécifier pour la conversion. 20
4.5 ($\mathcal{D}_{Acc}$) en représentation graphique 26
4.6 Représentation AVM suggérée par l'attribut *href* 26
4.7 Schéma récapitulatif du package 29

Table des matières

Résumé 2
Remerciements 2

1 Les structures de traits **3**
1.1 Définition 3
1.2 Les différents noms donnés à une structure de traits 3
1.3 Les boîtes de coïndexation 4
1.4 Graphes et structures de traits 4
1.5 Les différentes fonctions des structures de traits 5
1.5.1 Fonction informative 5
1.5.2 Fonction grammaticale 5
1.6 Opérations sur les structures de traits 6
1.6.1 La subsomption 6
1.6.2 L'unification 6
1.7 Structures de traits particulières 7
1.7.1 Les variables 7
1.7.2 Les listes 8

2 Package Feature Structure **9**
2.1 Présentation 9
2.2 Utilisation 9
2.3 Résumé des méthodes principales 10
2.3.1 Constructeur 10
2.3.2 Méthodes d'opérations sur les structures de traits 10

3 TermesParser **13**
3.1 Définition des unités lexicales reconnues 13
3.2 Règles de réécriture 14
3.3 Exemple de fichier 15

4 XmlParser **17**
4.1 Définition de la grammaire du parser 17
4.1.1 Définition de Type de Document 17
4.1.2 Schéma XML. 19
4.2 Génération du parser 21

4.2.1 XmlParser . . . 22
4.2.2 XmlBuilder . . . 22
4.3 Utilisation du parser XML dans d'autres grammaires . . . 23
4.4 Discussion sur la DTD . . . 24
4.4.1 Justification de l'attribut *label* . . . 24
4.4.2 Propositions d'améliorations de la DTD . . . 27

A Sources XML **31**
A.1 Document Type Definition . . . 31
A.2 Schéma XML . . . 34
A.3 Fichier de bindings pour JAXB . . . 42

B Sources Java **43**
B.1 TermesParser.jj . . . 43
B.2 Termes2FsXML.java . . . 52
B.3 FsXML2Termes.java . . . 54

C Ressources diverses **57**
C.1 Ressources logicielles . . . 57
C.2 Ressources en ligne . . . 57
C.2.1 Ressources Java . . . 57
C.2.2 Ressources XML . . . 58
C.2.3 Ressources sur les structures de traits . . . 58

Références bibliographiques **59**

Table des figures **60**

Table des matières **62**