Documentation
sur la réalisation de texte en français
avec SimpleNLG-EnFr 1.1

Par

Pierre-Luc Vaudry

Laboratoire de recherche appliquée en linguistique informatique

Département d'informatique et de recherche opérationnelle

Université de Montréal

Juillet 2012

Table des matières

1 Introduction

Cette documentation porte sur SimpleNLG-EnFr, adapté par Pierre-Luc Vaudry à partir de SimpleNLG v4.2. Alors que SimpleNLG permet de réaliser du texte en anglais, SimpleNLG-EnFr permet de le faire aussi bien en français qu'en anglais. Cette documentation a été mise à jour pour la version 1.1 de SimpleNLG-EnFr.

1.1 Qu'est-ce que SimpleNLG?

SimpleNLG est une bibliothèque Java pour la réalisation de texte en anglais. Elle a été développée à l'origine par Ehud Reiter, Albert Gatt et Dave Westwater, de l'université d'Aberdeen. La réalisation est la dernière étape dans le processus de la génération de texte et SimpleNLG est donc utile aux programmeurs écrivant une application dans ce domaine. SimpleNLG effectue la réalisation de surface d'un texte à partir d'une représentation abstraite où les structure syntaxiques et les unités lexicales ont déjà été déterminées. La figure 1 représente le rôle de SimpleNLG dans la génération de texte.

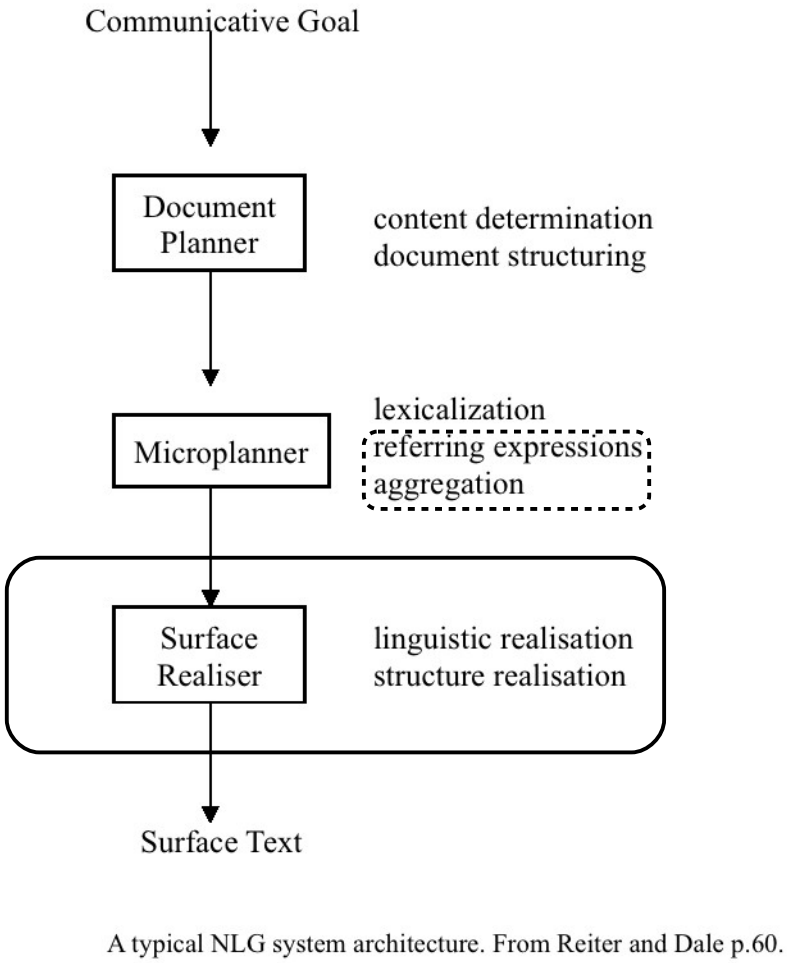

Figure 1: Rôle de SimpleNLG dans la génération de texte

(tiré de http://code.google.com/p/simplenlg/wiki/Tutorial)

Pour utiliser SimpleNLG, il faut construire dans le langage de programmation Java une structure de données représentant le texte à réaliser. Cette structure prend la forme d'un arbre composé d'objets appartenant à une des classes dérivées de NLGElement (voir figure 2, p. 7). Le réalisateur de SimpleNLG peut alors effectuer la transformation de cette structure de données en un texte formaté. SimpleNLG effectue alors automatiquement la flexion et la dérivation de mots, le placement des mots dans le bon ordre, la création des auxiliaires, la gestion de l'accord, de la ponctuation, de l'espacement, de la casse, etc. SimpleNLG peut aussi faire sur demande le remplacement d'un syntagme nominal (groupe nominal) par un pronom personnel et l'agrégation de propositions ayant, par exemple, le même sujet, mais des verbes différents.

Le tutoriel de la version originale de SimpleNLG[1] est une bonne introduction à l'approche générale de SimpleNLG. Il est conseillé de le consulter. Il y a peu de changements à apporter pour adapter le code des exemples de ce tutoriel à SimpleNLG-EnFr. Voir à ce sujet l'annexe 1 (p.63).

1.2 SimpleNLG-EnFr

1.2.1 L'adaptation de SimpleNLG pour le français

L'adaptation de SimpleNLG pour qu'il puisse faire la réalisation aussi bien en français qu'en anglais avait plusieurs objectifs :

- Produire du texte dans un français grammaticalement correct.
- Faire le moins de changements possible, d'une part, à l'interface programmeur (API) et d'autre part, à la structure interne et au code de SimpleNLG.
- Réutiliser la batterie de tests `JUnit`[2] déjà existante pour conserver les capacités de SimpleNLG intactes pour l'anglais et pour gagner en efficacité dans le débogage de la partie française de SimpleNLG-EnFr.
- Pouvoir réaliser du texte dans les deux langues dans le même document.
- Faciliter la tâche aux éventuelles adaptations futures de SimpleNLG-EnFr pour l'ajout d'autres langues, en séparant le plus possible ce qui était spécifique au français ou à l'anglais de ce qui était plus générique.

La conception de SimpleNLG-EnFr a posé plusieurs défis. Tout d'abord, il a fallu réorganiser certaines parties du code pour pouvoir isoler ce qui était spécifique à l'anglais de ce qui était commun au français et à l'anglais. Ensuite, des règles grammaticales et un lexique français ont été ajoutés, avec pour principales références grammaticales Grevisse (1993) et Mansouri (1996).

La grammaire française a posé certaines difficultés que ne posait pas la grammaire anglaise. Du côté de la syntaxe du groupe nominal, il a fallu déterminer si l'adjectif devait se placer devant ou derrière le nom et accorder le déterminant et l'adjectif avec celui-ci. Au niveau de la proposition, on peut nommer le choix des auxiliaires, la négation, l'accord du participe passé et le placement des pronoms compléments verbaux clitiques. Non seulement la négation comporte la plupart du temps deux adverbes à positionner, mais il faut pouvoir remplacer au besoin *pas* par *plus*, *point*, *aucunement*, etc. Quant au placement des pronoms verbaux clitiques (*me*, *te*, *le*, *lui*, *en*, *y*, etc.), il obéit à des règles complexes. Les propositions relatives (choix du pronom relatif, accord du verbe) ont aussi été un défi.

1 http://code.google.com/p/simplenlg/wiki/Tutorial

2 `JUnit` est une bibliothèque de test unitaire pour le langage de programmation Java.

En ce qui concerne la morphologie, il a fallu programmer les règles de féminisation des adjectifs et de mise au pluriel des noms et adjectifs. Il a fallu faire de même avec les verbes, dont la flexion est beaucoup plus variée qu'en anglais. Le choix des formes des pronoms personnels a aussi représenté un certain défi.

Un niveau supplémentaire de traitement, celui de la morphophonologie, a dû être ajouté. Ce type de règle est en effet plus fréquent en français qu'en anglais, du moins à l'écrit. Il implique l'interaction de mots contigus. Il s'agit par exemple de l'élision *que → qu'*, de la contraction *à + le → au* ou de la liaison *beau → bel*. En anglais, on n'avait que la règle *a → an* devant voyelle, qui était alors appliquée par le module morphologique. En français, ce type de règle est complexifiée par la question du « *h* aspiré ».

Quelques différences ont aussi été rencontrées au niveau de la ponctuation. Par exemple, on met une virgule devant la conjonction de coordination *mais*, mais pas devant *et*. En français, on fait en outre plus souvent suivre d'une virgule un syntagme adverbial placé en début de phrase.

Un lexique français par défaut, comprenant 3871 entrées, a été constitué pour SimpleNLG-EnFr. La première partie a été construite manuellement. Elle comprend les verbes auxiliaires *avoir* et *être*, ainsi que les déterminants, pronoms, prépositions, adverbes et conjonctions de subordination et de coordination les plus essentiels, en plus de quelques noms et adjectifs. La deuxième partie a été bâtie à partir de la liste de l'échelle Dubois Buyse, dont on a extrait les noms, les verbes, les adverbes, les adjectifs et les prépositions. Cette liste a été trouvée en format électronique sur la page web de Bacquet[3]. Les entrées correspondant aux noms, adjectifs et verbes de cette liste ont été enrichies en puisant dans le lexique ouvert de formes fléchies du français Morphalou 2.0 (CNRTL). Le format utilisé pour le lexique construit est le même que pour le lexique XML (voir p. 12.) anglais par défaut fourni avec SimpleNLG v4.2. Il y a toutefois de nouvelles balises, correspondant à de nouveaux traits (voir p. 7) ajoutés pour le français.

Certaines parties du code source de SimpleNLG ont dû être réorganisées pour pouvoir le rendre bilingue. On a donc dû effectuer la modification de certaines classes et l'ajout de nouvelles classes et *packages*. Par exemple, le code des modules correspondant aux niveaux de traitement syntaxique, morphologique et orthographique a dû être redistribué. On ne pouvait créer de sous-classes des anciennes classes contenant les règles de réalisation, car leurs fonctions étaient statiques (`static`) et/ou de visibilité privée (`private`). Le code ne dépendant que des sous-classes de `NLGElement` (voir figure 2, p. 7) a été ajouté dans ces dernières. Le code plus spécifique au traitement linguistique mais commun au français et à l'anglais a été placé dans de nouvelles classes abstraites et le reste a été mis dans de nouvelles classes. Les fonctions de ces nouvelles classes ont été mises non statiques (`static`) et de visibilité publique (`public`) ou protégée (`protected`). De plus, des fonctions dans les classes `WordElement`, `PhraseElement` et `StringElement` peuvent maintenant déterminer dynamiquement la classe dont il faut utiliser les fonctions pour le traitement de chaque élément, selon la langue de celui-ci.

Aussi, certaines fonctions ont dû être créées dans le lexique pour fournir, par exemple, la conjonction de coordination par défaut dans la classe `CoordinatedPhraseElement`. Auparavant, il y avait simplement dans cette classe une chaîne de caractères littérale *"and"*.

SimpleNLG v4.2 est distribué sous la Mozilla Public License. Cela a permis son adaptation et la distribution (sous la même licence) de SimpleNLG-EnFr. Pour plus de détails, voir l'annexe 3 (p. 65).

3 http://o.bacquet.free.fr/db2.htm

1.2.2 Utilisation de SimpleNLG-EnFr

Dans SimpleNLG-EnFr, la langue des unités lexicales est donnée par le lexique d'où ils proviennent et la langue des syntagmes et des unités plus grandes est donnée par la *factory* avec laquelle ils ont été créés. La langue d'une *factory* est elle-même donnée par le lexique avec laquelle elle a été créée. Dans la version bilingue, une *factory* ne peut donc pas avoir un lexique `null`; le lexique par défaut anglais est attribué le cas échéant. Sachant cela, **on peut combiner des éléments `NLGElement` ayant des langues différentes** (provenant de *factories* ou lexiques différents) et SimpleNLG-EnFr les traitera de la manière la plus appropriée, appliquant la morphologie française aux mots français et la morphologie anglaise aux mots anglais, par exemple, et ceci de façon transparente. À la différence de la version originale de SimpleNLG, la version bilingue pourrait ainsi potentiellement gérer un grand nombre de langues et il y serait moins difficile d'ajouter des extensions pour l'espagnol, l'allemand, l'italien, etc.

Ce document détaille ce qu'il faut savoir pour effectuer la réalisation de texte en français avec SimpleNLG-EnFr. On y décrit comment utiliser chaque type d'unité lexicale, de syntagme et d'élément de document : quels traits (*features*) sont pertinents, quelles méthodes utiliser pour obtenir tel ou tel point de grammaire, etc. Il contient quelques exemples d'utilisation des capacités de SimpleNLG. Pour davantage d'exemples, il est suggéré de consulter les fichiers des *packages* `simplenlg.test.english`, `simplenlg.test.french` et `simplenlg.examples`.

Pour avoir une première idée de la façon d'utiliser SimpleNLG-EnFr, l'exemple 1.1, à la page suivante, montre les déclarations d'importation, le chargement du lexique, de la *factory* et du réalisateur, ainsi que la construction et la réalisation d'une phrase simple en français. Tous les exemples de ce document peuvent être trouvés dans le *package* `simplenlg.examples` ou dans la classe de test JUnit `simplenlg.test.french.DocumentationTest`.

Exemple 1.1: Construction et réalisation d'une phrase simple.

```
import simplenlg.framework.*;
import simplenlg.lexicon.Lexicon;
import simplenlg.lexicon.french.XMLLexicon;
import simplenlg.realiser.*;
import simplenlg.phrasespec.*;

public class GreetCrowdFr {
     public static void main(String[] args) {

          Lexicon lexicon = new XMLLexicon();
          NLGFactory factory = new NLGFactory(lexicon);
          Realiser realiser = new Realiser();

          NPPhraseSpec theMan = factory.createNounPhrase("le", "homme");
          NPPhraseSpec theCrowd = factory.createNounPhrase("le", "foule");

          SPhraseSpec greeting =
               factory.createClause(theMan, "saluer", theCrowd);

          String outString = realiser.realiseSentence(greeting);
          System.out.println(outString);
     }
}
```

Sortie :

```
L'homme salue la foule.
```

La couverture grammaticale de la partie anglaise de SimpleNLG-EnFr correspond exactement à celle de SimpleNLG v4.2. Pour ce qui est de la partie française, nous avons tenté de satisfaire au minimum aux recommandations du *Français fondamental (1er Degré)*. Dans ce but, dans la version 1.1, les notions grammaticales suivantes ont été ajoutées :

- les types de propositions interrogatives correspondants à ceux déjà disponibles en anglais (formulées avec l'introducteur *est-ce que*)
- un mécanisme pour la formation de propositions relatives, unique à la partie française
- la conjugaison du conditionnel (présent et passé) et du subjonctif (présent, passé et passé surcomposé)
- la règle pour le futur et le conditionnel avec le *si* de condition
- l'adjectif ordinal : positionnement avant le nom et absence d'élision de l'article

Il restera encore à ajouter dans une version ultérieure le traitement complet de la représentation des nombres en adjectifs cardinaux et ordinaux. L'ajout de l'indicatif passé simple et passé antérieur et du subjonctif imparfait et plus-que-parfait n'est pas envisagé, leur utilité pratique n'étant pas jugée suffisante. De même, les propositions interrogatives avec inversion du sujet n'ont pas été incluses.

Toutes les classes qui font partie de la chaîne de traitement de SimpleNLG dérivent de la classe `NLGElement` et sont contenues dans le *package* `simplenlg.framework`. Ces classes sont représentés dans la figure 2.

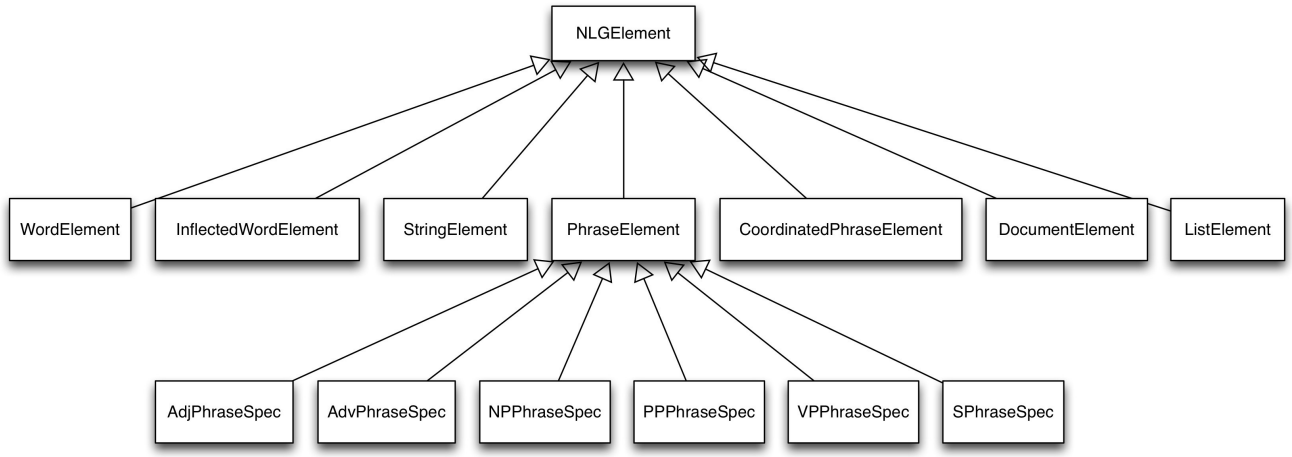

Figure 2: Diagramme d'héritage de la classe `NLGElement`

Les objets de classe `NLGElement` ont tous une catégorie grammaticale (`category`) et un ensemble de traits (*features*) associant un nom de trait (`String`) à un objet de classe variable (`Object`). Les classes contenues dans les *packages* `simplenlg.features` et `simplenlg.features.french` procurent une référence aisée au nom de tous ces traits (voir figure 3). Les commentaires qui les accompagnent décrivent sommairement leur utilisation et le type d'objet attendu comme valeur de chaque trait. L'utilisateur de SimpleNLG peut à volonté accéder à ces traits en cours d'exécution et même en créer de nouveaux, si cela s'avérait nécessaire. On utilise les méthodes suivantes pour accéder aux traits d'un élément :

- `setFeature(String` nomDeTrait, {`Object` ou type primitif, sauf `char`} valeur)
- `Object getFeature(String` nomDeTrait)
- `Boolean getFeatureAsBoolean(String` nomDeTrait)
- `Integer getFeatureAsInteger( String` nomDeTrait)
- (même chose pour `Double`, `Float` et `Long`)
- `String getFeatureAsString( String` nomDeTrait)
- `List<String> getFeatureAsStringList( String` nomDeTrait)
- `NLGElement getFeatureAsElement( String` nomDeTrait)
- `List<NLGElement> getFeatureAsElementList( String` nomDeTrait)

Les méthodes qui retournent une liste ne retournent pas la liste originale, mais une copie de celle-ci. La valeur par défaut d'un trait booléen est `false` et celle de la plupart des autres traits est `null`.

La figure 3 montre le contenu du *package* `simplenlg.features`. Les formes arrondies à double bordure sont les classes contenant les constantes `String` représentant le nom des traits. Celles qui n'ont qu'une simple bordure sont des types énumérés qui sont utilisés pour la valeur de certains traits. Le *package* `simplenlg.features.french` a été créé pour SimpleNLG-EnFr.

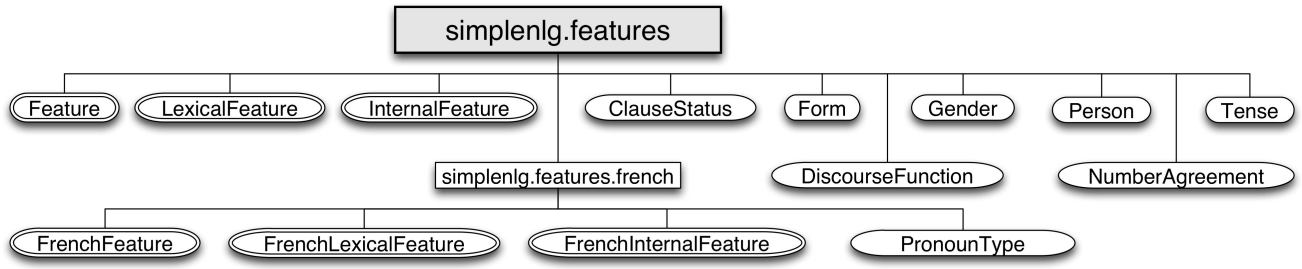

Figure 3: Contenu du package *`simplenlg.features`*

L'exemple 1.2 (p. 9) illustre l'utilisation des traits généraux et spécifiques au français, ainsi que la recherche d'un mot dans le lexique. La plupart des traits qui ont été créés spécialement pour la partie française de SimpleNLG-EnFr sont des traits utilisés principalement par le lexique, contrairement au trait apparaissant dans l'exemple 1.2.

Notez que les instructions `import` et les déclarations du lexique, de la *factory* et du réalisateur ne seront pas reprises dans le reste des exemples de ce document, mais seront implicites.

*Exemple 1.2: Utilisation des traits (*features*) et du lexique*

```
import simplenlg.framework.*;
import simplenlg.lexicon.Lexicon;
import simplenlg.lexicon.french.XMLLexicon;
import simplenlg.realiser.*;
import simplenlg.phrasespec.*;

import simplenlg.features.*;
import simplenlg.features.french.*;

public class SaluerPlusFoule {

     public static void main(String[] args) {

          Lexicon lexicon = new XMLLexicon();
          NLGFactory factory = new NLGFactory(lexicon);
          Realiser realiser = new Realiser();

          NPPhraseSpec theMan = factory.createNounPhrase("le", "homme");
          NPPhraseSpec theCrowd = factory.createNounPhrase("le", "foule");

          SPhraseSpec greeting =
               factory.createClause(theMan, "saluer", theCrowd);

          greeting.setFeature(Feature.NEGATED, true);

          String outString = realiser.realiseSentence(greeting);
          System.out.println(outString);

          WordElement plus =
               lexicon.lookupWord("plus", LexicalCategory.ADVERB);
          greeting.setFeature(FrenchFeature.NEGATION_AUXILIARY, plus);

          outString = realiser.realiseSentence(greeting);
          System.out.println(outString);
     }

}
```

Sortie :

```
L'homme ne salue pas la foule.
L'homme ne salue plus la foule.
```

2 Unités lexicales

Les unités lexicales sont stockées dans des objets de la classe `WordElement`. La classe `WordElement` est dérivée de la classe `NLGElement`. Il faut souligner que les `WordElement` sont uniques, c'est-à-dire que toutes les structures syntaxiques comprenant le nom commun féminin *table* réfèrent à la même instance de `WordElement` que celle référencée par le lexique d'où *table* provient.

2.1 Le lexique

Les unités lexicales font normalement partie d'un lexique représenté par un objet d'une classe dérivée de la classe abstraite `simplenlg.lexicon.Lexicon`. Ces classes sont représentées dans la figure 4. Les boîtes dont la bordure est tiretée représentent des classes qui ont été créées pour SimpleNLG-EnFr. Pour l'instant, pour le français, il n'y a qu'une classe disponible pour charger un lexique individuel : `simplenlg.lexicon.french.XMLLexicon`. La classe `NIHDBLexicon` est une interface spécifique pour une banque de données lexicale spécialisée de langue anglaise. La classe `MultipleLexicon` sert à charger plus d'un lexique à la fois. Les lexiques ainsi chargés sont consultés dans l'ordre dans lequel ils ont été ajoutés.

Le fichier XML contenant les données du lexique français par défaut est situé dans le même répertoire que le fichier java de la classe correspondante, soit :

`src/simplenlg/lexicon/french/default-french-lexicon.xml`.

Une copie de ce fichier se trouve également dans le répertoire `res`. Cette copie n'est pas utilisée par le programme, elle n'est là que pour la convenance de l'utilisateur. Le même principe s'applique pour le lexique par défaut anglais.

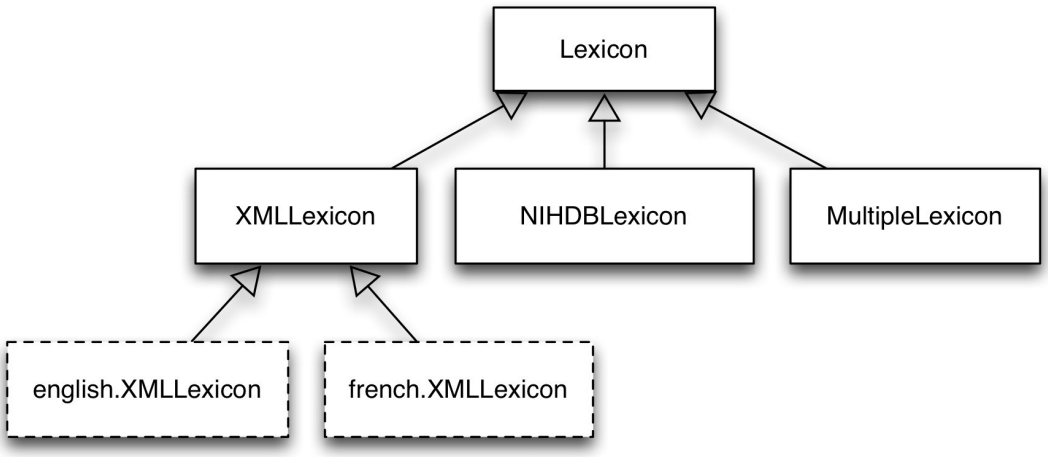

Figure 4: Diagramme d'héritage de la classe `Lexicon`

Il y a plusieurs façons d'accéder à un `WordElement` à partir d'une forme donnée. On peut interroger directement le lexique grâce aux méthodes suivantes (pour l'`id`, voir page 12):

Nom de la méthode	Argument(s)	Type retourné	Description
`getWords`	Forme de base Catégorie (optionnelle)	`List<WordElement>`	Retournent tous les mots correspondant à (aux) argument(s) donnés.
`getWordsFromVariant`	Variante (forme fléchie) Catégorie (optionnelle)		
`getWordsByID`	`id`		
`getWord`	Forme de base Catégorie (optionnelle)	`WordElement`	• Retournent le premier mot de la liste retournée par le getWords... (au pluriel) correspondant. • Si la liste est vide, un nouveau mot est créé et retourné.
`getWordFromVariant`	Variante (forme fléchie) Catégorie (optionnelle)		
`getWordByID`	`id`		
`hasWord`	Forme de base Catégorie (optionnelle)	`boolean`	Vérifient si le mot correspondant existe.
`hasWordFromVariant`	Variante (forme fléchie) Catégorie (optionnelle)		
`hasWordByID`	`id`		
`lookupWord`	Forme de base, variante ou `id` Catégorie (optionnelle)	`WordElement`	• Cherche successivement par forme de base, par variante, puis par `id`. Retourne le premier mot trouvé. • Si aucun mot n'est trouvé, un nouveau mot est créé et retourné.

• Nouveau dans SimpleNLG-EnFr :

`getWords`	Catégorie Ensemble (*map*) de traits	`List<WordElement>`	• Cherchent les mots possédant la bonne catégorie et toutes les paires <trait, valeur> fournies. • Aucun mot n'est créé.
`getWord`		`WordElement`	
`hasWord`		`boolean`	

On peut aussi passer par la *factory*. On utilise alors les méthodes `createNLGElement(...)` ou `createWord(...)`.

Dans un lexique multiple (`MultipleLexicon`), quand un mot n'est pas trouvé dans le premier

lexique, il est recherché dans le deuxième lexique, et ainsi de suite. Dès que des mots sont trouvés dans un des lexiques, la recherche s'arrête et s'il existe des doublons dans un lexique subséquent, il ne seront pas repêchés. Pour qu'ils le soient, il faut d'abord appeler la méthode `setAlwaysSearchAll(…)`, qui permet de demander une recherche dans tous les lexiques.

Quand un mot n'est pas trouvé, un nouveau `WordElement` est créé et retourné si la recherche a été faite avec `lookupWord` ou `getWord`... avec forme de base, variante ou ID. Avec un `XMLLexicon`, dans SimpleNLG-EnFr, le nouveau mot est également ajouté au lexique chargé en mémoire. De cette façon, si l'utilisateur ajoute des traits à ce nouveau mot, ils seront présents chaque fois que le mot sera recherché de nouveau pendant cette exécution du programme. Avec un lexique multiple, le mot sera créé en utilisant la méthode correspondante dans le premier lexique de la liste. Si la liste est vide, un nouveau `WordElement` sera créé et retourné, mais ne sera pas conservé en mémoire. Un `WordElement` se voit toujours attribuer une référence au lexique qui l'a créé, ce qui permet entre autres de déterminer sa langue. En effet, dans SimpleNLG-EnFr, tout lexique est soit de langue française, soit de langue anglaise.

2.2 Caractéristiques et représentation des unités lexicales

Les unités lexicales (`WordElement`) ont toutes une catégorie grammaticale de type énuméré `LexicalCategory`. (La valeur de la catégorie selon ce type énuméré est placée entre parenthèses dans les sous-titres de cette section.) Les traits lexicaux qui les caractérisent varient selon cette catégorie. Ces traits sont normalement stockés dans le fichier XML représentant le lexique, mais peuvent aussi être modifiés par l'utilisateur en cours d'exécution. Les unités lexicales ont également une propriété `baseForm`, qui représente la forme de base de l'unité lexicale (masculin singulier pour un nom, adjectif ou article; infinitif pour un verbe; etc.). Ils ont de plus une propriété dont l'utilisation est optionnelle, `id`, qui sert à identifier de façon unique l'unité lexicale si besoin est.

Le format du fichier XML pouvant être chargé par la classe `XMLLexicon` est illustré dans l'exemple 2.1, tiré du lexique français par défaut. Chaque balise `<Word>` correspond à un `WordElement`. Les valeurs de la forme de base, de la catégorie et du `id` sont écrites entre les balises correspondantes. Il en est de même pour les traits dont la valeur est une chaîne de caractères. La valeur d'un type énuméré est quant à elle représentée par la chaîne de caractères correspondante en minuscules. Les traits dont la valeur est booléenne apparaissent comme des balises sans contenu quand leur valeur est `true` ou pas du tout si elle est `false`. Le nom d'une balise de trait correspond à la valeur de la constante correspondante. Par exemple, dans la classe `simplenlg.features.french .FrenchLexicalFeature`, on a la définition suivante du trait français d'élision :

```
public static final String VOWEL_ELISION = "vowel_elision";
```

Rappelons qu'un trait (*feature*, voir p. 7) est l'association d'un `String` et d'un `Object.` La première entrée de l'exemple 2.1, l'article défini *le*, a été rédigée à la main et l'`id` correspond à celui de l'entrée du déterminant *the* du lexique XML anglais par défaut. La deuxième entrée de l'exemple 2.1 a été générée en partie automatiquement. En effet, les noms, adjectifs et adverbes de la liste de l'échelle orthographique Dubois Buyse, trouvée sur la page web de Bacquet[4], ont été complétés par les formes fléchies de Morphalou 2.0 (CNRTL) (voir p. 4). Quelques traits correspondant à ces formes fléchies sont pour cette raison redondants, car SimpleNLG-EnFr peut les calculer si elles sont relativement régulières (voir p. 15, 16 et 19). L'`id` des entrées dont les formes fléchies ont été tirées de Morphalou 2.0 provient également de celle-ci.

4 http://o.bacquet.free.fr/db2.htm

Exemple 2.1: Entrées lexicales tirées du lexique XML français par défaut.

```
<lexicon>

  <word>
    <base>le</base>
    <category>determiner</category>
    <id>E0060487</id>
    <feminine_singular>la</feminine_singular>
    <plural>les</plural>
    <vowel_elision/>
  </word>

  <word>
    <base>devenir</base>
    <category>verb</category>
    <id>devenir_2</id>
    <copular/>
    <auxiliary_etre/>
    <presentParticiple>devenant</presentParticiple>
    <present2p>devenez</present2p>
    <imperative2p>devenez</imperative2p>
    <imparfait_radical>deven</imparfait_radical>
    <present1p>devenons</present1p>
    <imperative1p>devenons</imperative1p>
    <future_radical>deviendr</future_radical>
    <present3p>deviennent</present3p>
    <present1s>deviens</present1s>
    <present2s>deviens</present2s>
    <imperative2s>deviens</imperative2s>
    <present3s>devient</present3s>
    <pastParticiple>devenu</pastParticiple>
    <feminine_past_participle>devenue</feminine_past_participle>
  </word>

</lexicon>
```

Il est possible s'utiliser SimpleNLG simplement pour effectuer la flexion de mots isolés. Dans ce cas, on peut spécifier la valeur des traits flexionnels, indiqués dans les sous-sections 2.3 à 2.7 pour chaque catégorie grammaticale. Il faut alors créer un `InflectedWordElement` à partir du `WordElement` désiré et lui ajouter les traits requis pour la flexion. (Remarquez qu'il faut procéder de la même façon pour ajouter le trait `ELIDED` à une instance d'une unité lexicale (voir page 48).) Alternativement, on peut construire un syntagme de la bonne catégorie, régler ses traits et lui assigner la tête désirée. Ensuite, on utilise le réalisateur pour obtenir la forme fléchie. L'exemple 2.2 montre comment obtenir une forme fléchie d'une unité lexicale. Voir les pages 18 et 19 pour la description des traits utilisés ici. Dans cet exemple et dans plusieurs autres qui suivront, la fonction `outln(NLGElement)` est utilisée pour ne pas alourdir le code. Sa définition se trouve à l'annexe 2 (p. 64).

Exemple 2.2: Flexion d'une unité lexicale isolée avec `InflectedWordElement` *et avec syntagme.*

```
WordElement le = lexicon.getWord("le", LexicalCategory.DETERMINER);
InflectedWordElement leInfl = new InflectedWordElement(le);
leInfl.setFeature(LexicalFeature.GENDER, Gender.FEMININE);
outln(leInfl);

VPPhraseSpec devenirInfl = factory.createVerbPhrase("devenir");
devenirInfl.setFeature(Feature.PERSON, Person.SECOND);
devenirInfl.setFeature(Feature.NUMBER, NumberAgreement.PLURAL);
devenirInfl.setFeature(Feature.TENSE, Tense.FUTURE);
outln(devenirInfl);
```

Sortie :

```
la
deviendrez
```

Il n'est habituellement pas nécessaire de manipuler les traits flexionnels, du moins au niveau des mots individuels. Certains peuvent être manipulés au niveau du syntagme; voir la section 3 (p. 22) à ce sujet.

Certains traits lexicaux s'appliquent potentiellement à tous les `WordElement` en français :

Trait	Classe du trait	Valeur	Description
`VOWEL_ELISION`	`FrenchLexicalFeature`	`Gender`	Mot dont la voyelle finale s'élide devant une voyelle ou un *h* qui n'est pas « aspiré »
`ASPIRED_H`	`FrenchLexicalFeature`	`String`	Commence par un « *h* aspiré »
`NE_ONLY_NEGATION`	`FrenchLexicalFeature`	`Boolean`	`true` si l'unité lexicale provoque une négation avec l'adverbe de négation *ne* seulement

Un mot dont la voyelle finale est susceptible de s'élider (ayant le trait `VOWEL_ELISION`) subira cette élision si le mot suivant commence (graphiquement) par une voyelle ou la lettre h, n'est pas un adjectif ordinal ou apparenté (suffixe *-ième*) et ne possède pas le trait `ASPIRED_H`. Attention, un mot peut posséder ce dernier trait sans pour autant commencer par la lettre *h*. (ex. : *oui*) Quand l'élision a lieu, une apostrophe remplace la voyelle élidée et aucun espace ne suit le mot dont la voyelle finale est élidée. La liste des mots faisant exception à la règle de l'élision et à d'autres phénomènes morphophonologiques similaires a été prise dans Grevisse (1993 : §47-50, p. 62-75).

La négation avec l'adverbe de négation *ne* seulement (sans auxiliaire de négation) se produit quand l'unité lexicale qui possède le trait `NE_ONLY_NEGATION` est située dans un syntagme nominal en position de sujet ou de complément d'un verbe (par opposition à un modificateur). Par exemple, le pronom indéfini *personne*, le déterminant indéfini *aucun* et la conjonction de coordination *ni* possèdent ce trait. Par contre, si l'auxiliaire de négation a été changé pour *plus*, il est tout de même réalisé.

Les sous-sections suivantes traitent de chaque catégorie d'unité lexicale. La valeur de la catégorie selon le type énuméré `LexicalCategory` est placée entre parenthèses dans chaque sous-titre.

2.3 Nom (NOUN)

Traits lexicaux :

Trait	Classe du trait	Valeur	Description
GENDER	LexicalFeature	Gender	Genre du nom; masculin si non spécifié.
PLURAL	LexicalFeature	String	Forme plurielle, si irrégulière.
PROPER	LexicalFeature	Boolean	Nom propre; ne sera pas mis au pluriel.
OPPOSITE_GENDER	FrenchLexicalFeature	String	Nom de genre opposé, s'il y a lieu.

Traits flexionnels :

Trait	Classe du trait	Valeur	Description
NUMBER	Feature	NumberAgreement	Nombre du syntagme nominal.

Flexion :

Le pluriel des noms, quand il n'est pas spécifié par le trait `LexicalFeature.PLURAL`, est formé (comme celui des adjectifs) en suivant les règles suivantes (Grevisse, 1993 : §504-505, p. 538-539) :

- forme de base se terminant en *s*, *x* ou *z* →aucun changement
- forme de base se terminant en *au* ou *eu* → ajout d'un suffixe *x*
- forme de base se terminant en *al* → terminaison devient *aux*
- le reste → ajout d'un suffixe *s*

2.4 Pronom (PRONOUN)

Traits lexicaux :

Trait	Classe du trait	Valeur	Description
PERSON	Feature	Person	Personne du pronom.
NUMBER	Feature	NumberAgreement	Nombre du pronom.
GENDER	LexicalFeature	Gender	Genre du pronom.
REFLEXIVE	LexicalFeature	Boolean	`true` s'il s'agit d'une forme pronominale réfléchie.
DETACHED	FrenchLexicalFeature	Boolean	`true` si cette forme pronominale est une forme normalement disjointe du verbe (ex. : *moi*)
PRONOUN_TYPE	FrenchLexicalFeature	PronounType	Type du pronom (personnel, démonstratif, relatif, etc.)

Flexion :

Les formes pronominales ont été entrées individuellement dans le lexique, contrairement aux formes féminines et plurielles de l'adjectif, par exemple. Elles sont sélectionnées selon la fonction syntaxique qu'occupe le pronom et si celui-ci est disjoint du verbe (Grevisse, 1993 : §633-634, p. 1006-1008).

2.5 Adjectif (`ADJECTIVE`)

Traits lexicaux :

Trait	Classe du trait	Valeur	Description
`COMPARATIVE`	`LexicalFeature`	`String`	Forme comparative synthétique, s'il y a lieu.
`PLURAL`	`LexicalFeature`	`String`	Forme du masculin pluriel, si c'est une forme irrégulière.
`FEMININE_SINGULAR`	`FrenchLexicalFeature`	`String`	Forme du féminin singulier, si c'est une forme irrégulière.
`FEMININE_PLURAL`	`FrenchLexicalFeature`	`String`	Forme du féminin pluriel, si c'est une forme irrégulière.
`LIAISON`	`FrenchLexicalFeature`	`String`	Forme de liaison, s'il y a lieu : forme que prend un adjectif au masculin singulier dans les mêmes conditions que l'élision mentionnée ci-haut. (ex.: *beau→bel*)
`PREPOSED`	`FrenchLexicalFeature`	`Boolean`	`true` si l'adjectif est antéposé par défaut, plutôt que postposé. (ex.: *beau*)

Traits flexionnels :

Trait	Classe du trait	Valeur	Description
`NUMBER`	`Feature`	`NumberAgreement`	Nombre du syntagme nominal associé.
`GENDER`	`LexicalFeature`	`Gender`	Genre du nom associé.

Flexion :

La forme féminine des adjectifs, quand elle n'est pas spécifiée par le trait `FrenchLexicalFeature.FEMININE_SINGULAR`, est formée en suivant les règles suivantes, dans l'ordre (Grevisse, 1993 : §528-534, p. 866-873) :

- forme de base se terminant en *e* →aucun changement
- forme de base se terminant en *el* ou *eil* → ajout d'un suffixe *le*

- forme de base se terminant en *en* ou *on* → ajout d'un suffixe n*e*
- forme de base se terminant en *et* → ajout d'un suffixe *te*
- forme de base se terminant en *eux* → terminaison devient *euse*
- forme de base se terminant en *er* → terminaison devient *ère*
- forme de base se terminant en *eau* → terminaison devient *elle*
- forme de base se terminant en *gu* → ajout d'un suffixe *ë*
- forme de base se terminant en *g* → ajout d'un suffixe *ue*
- forme de base se terminant en *eur* et en remplaçant cette terminaison par *ant* on obtient une variante d'une entrée figurant dans le lexique (le participe présent d'un verbe) → terminaison devient *euse*
- forme de base se terminant en *teur* → terminaison devient *trice*
- forme de base se terminant en *if* → terminaison devient *ive*
- le reste → ajout d'un suffixe *e*

Le pluriel des adjectifs, quand il n'est pas spécifié par le trait `LexicalFeature.PLURAL` ou par le trait `FrenchLexicalFeature.FEMININE_PLURAL`, selon le cas, est formé comme celui des noms, après la formation du féminin s'il y a lieu.

Trait dérivationnel :

Trait	Classe du trait	Valeur	Description
`IS_COMPARATIVE`	`Feature`	`Boolean`	Régler à `true` pour mettre l'adjectif sous forme comparative synthétique, si celle-ci existe.

Dérivation :

En anglais, le comparatif et le superlatif de l'adjectif peuvent être exprimés de manière synthétique (en un seul mot). En français, cela est possible seulement pour le comparatif de quelques adjectifs :

- *bon → meilleur*
- *mauvais → pire*
- *petit → moindre*

Ces formes comparatives synthétiques sont stockées dans le trait `COMPARATIVE` de ces trois adjectifs et on peut les obtenir à l'aide du trait `IS_COMPARATIVE` décrit ci-haut. On peut régler ce trait au niveau du syntagme adjectival (`AdjPhraseSpec`) ou d'un `InflectedWordElement.` Pour les formes comparative et superlative analytiques, voir la page 25 et l'exemple 4.1 (p. 48).

2.6 Déterminant (`DETERMINER`)

Traits lexicaux :

Trait	Classe du trait	Valeur	Description
`FEMININE_SINGULAR`	`FrenchLexicalFeature`	`String`	Forme du féminin singulier.

Trait	Classe du trait	Valeur	Description
PLURAL	LexicalFeature	String	Forme du pluriel.
PARTICLE	Feature	String	Particule de certains déterminants démonstratifs (*ci* pour *ce -ci* et *là* pour *ce -là*).

Traits flexionnels :

Trait	Classe du trait	Valeur	Description
NUMBER	Feature	NumberAgreement	Nombre du syntagme nominal associé.
GENDER	LexicalFeature	Gender	Genre du nom associé.

Flexion :

La forme féminine et la forme plurielle (de genre indifférencié) des articles sont formées exclusivement en référence aux traits lexicaux de ceux-ci.

2.7 Verbe (VERB)

Traits lexicaux :

Trait	Classe du trait	Valeur	Description
PAST_PARTICIPLE	LexicalFeature	String	Forme participe passé, si c'est une forme irrégulière.
FEMININE_PAST_PARTICIPLE	FrenchLexicalFeature	String	Forme féminine du participe passé, si c'est une forme irrégulière.
PRESENT_PARTICIPLE	LexicalFeature	String	Forme participe présent, si c'est une forme irrégulière.
FUTURE_RADICAL	FrenchLexicalFeature	String	Radical du futur, s'il est irrégulier. Sert aussi pour le conditionnel
IMPARFAIT_RADICAL	FrenchLexicalFeature	String	Radical de l'imparfait, s'il est irrégulier.
IMPERATIVE2S IMPERATIVE1P IMPERATIVE2P	FrenchLexicalFeature	String	Formes de l'impératif présent (personne, nombre), si elles sont irrégulières.

Trait	Classe du trait	Valeur	Description
PRESENT1S PRESENT2S PRESENT3S PRESENT1P PRESENT2P PRESENT3P	FrenchLexicalFeature	String	Formes de l'indicatif présent (personne, nombre), si elles sont irrégulières.
SUBJUNCTIVE1S SUBJUNCTIVE2S SUBJUNCTIVE3S SUBJUNCTIVE1P SUBJUNCTIVE2P SUBJUNCTIVE3P	FrenchLexicalFeature	String	Formes du subjonctif présent (personne, nombre), si elles sont irrégulières.
CLITIC_RISING	FrenchLexicalFeature	Boolean	true si le verbe provoque la montée des pronoms compléments clitiques lorsqu'utilisé comme modal.
COPULAR	FrenchLexicalFeature	Boolean	true si le verbe est une copule et a donc un attribut du sujet au lieu d'un complément d'objet direct.

Dans le cas du trait COPULAR, on spécifie l'attribut du sujet comme si c'était un complément d'objet direct. Ceci dans le but de simplifier et de rester compatible avec SimpleNLG. L'accord avec le sujet est tout de même fait quand l'attribut est un adjectif

Traits flexionnels :

Trait	Classe du trait	Valeur	Description
PERSON	Feature	Person	Personne du sujet.
NUMBER	Feature	NumberAgreement	Nombre du sujet.
GENDER	LexicalFeature	Gender	Genre du sujet.
TENSE	Feature	Tense	Temps de l'indicatif (imparfait (PAST), présent, futur).
FORM	Feature	Form	Forme (participe présent, participe passé, indicatif (NORMAL), gérondif, impératif).

Flexion :

Si elle n'est pas spécifiée dans le lexique par les traits lexicaux appropriés, la flexion des verbes du premier groupe (se conjuguant comme *aimer*), du deuxième groupe (se conjuguant comme *finir* ou *haïr*) et de certains verbes du troisième groupe (se conjuguant comme *voir*, *vendre* ou *mettre*) est prise en charge par des règles morphologiques inspirées de Mansouri (1996) ou, pour le participe, tirées de

(Grevisse, 1993 : §777-778, p. 1213-1216). Certaines règles phonologiques sont appliquées au moment de l'ajout du suffixe au radical tels qu'ils ont été déterminés par ces règles (Grevisse, 1993 : §760-761). Les participes sont fléchis en genre et en nombre lorsqu'approprié.

2.8 Adverbe (ADVERB)

Trait lexical :

Trait	Classe du trait	Valeur	Description
COMPARATIVE	LexicalFeature	String	Forme comparative synthétique, s'il y a lieu.

Trait dérivationnel :

Trait	Classe du trait	Valeur	Description
IS_COMPARATIVE	Feature	Boolean	Régler à true pour mettre l'adverbe sous forme comparative synthétique, si celle-ci existe.

Dérivation :

En anglais, le comparatif et le superlatif de l'adverbe peuvent être exprimés de manière synthétique (en un seul mot). En français, cela est possible seulement pour le comparatif de quelques adverbes :

- *beaucoup → plus*
- *peu → moins*
- *bien → mieux*
- *mal → pis*

Ces formes comparatives synthétiques sont stockées dans le trait COMPARATIVE de ces quatre adverbes et on peut les obtenir à l'aide du trait IS_COMPARATIVE décrit ci-haut. On peut régler ce trait au niveau du syntagme adjectival (AdjPhraseSpec) ou d'un InflectedWordElement. Pour les formes comparative et superlative analytiques, voir la page 25.

2.9 Préposition (PREPOSITION)

Aucun trait particulier.

2.10 Conjonction de coordination (`CONJUNCTION`)

Traits lexicaux :

Trait	Classe du trait	Valeur	Description
`REPEATED_CONJUNCTION`	`FrenchLexicalFeature`	`Boolean`	`true` si la conjonction de coordination doit être répétée avant chaque élément coordonné (ex.: *ni*).
`NO_COMMA`	`FrenchLexicalFeature`	`Boolean`	`true` si la conjonction de coordination ne doit pas être précédée d'une virgule.
`NUMBER`	`Feature`	`NumberAgreement`	`PLURAL` si la conjonction de coordination doit provoquer la mise au pluriel du syntagme coordonné.

2.11 Conjonction de subordination (`COMPLEMENTISER`)

Cette catégorie est aussi appelée complémenteur. Aucun trait particulier.

3 Syntagmes (groupes de mots)

Les syntagmes sont représentés par les sous-classes de `PhraseElement` se trouvant dans le package `simplenlg.phrasespec`. Les syntagmes coordonnés sont quant à eux représentés par la classe `CoordinatedPhraseElement`. `PhraseElement` et `CoordinatedPhraseElement` sont toutes deux dérivées de la classe `NLGElement`. Se référer à la figure 2, page 7.

Tous les syntagmes peuvent avoir des compléments (`addComplement(...)`) et des modificateurs (`addModifier(...)`). Les modificateurs sont de deux types : les prémodificateurs (`addPreModifier(...)`) et les postmodificateurs (`addPostModifier(...)`). On peut laisser se faire automatiquement le choix entre ces deux derniers types de modificateurs en utilisant `addModifier(...)`. Les compléments sont placés juste après la tête du syntagme, tandis que les postmodificateurs viennent en dernier. Les prémodificateurs sont habituellement placés juste avant la tête du syntagme. Nous avons donc l'ordre suivant :

prémodificateurs, tête du syntagme, compléments, postmodificateurs

Les sous-sections suivantes décrivent chacune des catégories de syntagmes. La classe correspondante est spécifiée entre parenthèses dans chaque sous-titre.

3.1 Syntagme nominal (`NPPhraseSpec`)

On spécifie le déterminant et le nom ou le pronom d'un syntagme nominal soit dans l'appel de la méthode `createNounPhrase`(...) de la *factory*, soit avec les méthodes de `NPPhraseSpec` `setSpecifier`(...) et `setNoun`(...) ou `setPronoun`(...), respectivement.

Quand on ajoute un modificateur adjectival à un syntagme nominal avec la méthode `addModifier(...)`, il est placé par défaut avant ou après le nom en fonction de son trait `FrenchLexicalFeature.PREPOSED`. Les modificateurs non adjectivaux sont toujours placés après le nom. Il demeure possible de forcer l'une ou l'autre position avec `addPreModifier(...)` ou `addPostModifier(...)`.

Par défaut, un syntagme nominal est au singulier. On peut régler le nombre d'un syntagme nominal en attribuant une valeur au trait `Feature.NUMBER` ou en passant par la méthode `setPlural(boolean)`.

Pour créer une anaphore pronominale tout en conservant l'antécédent tel quel, on copie le syntagme nominal à l'aide de `createNounPhrase`(...) et on ajoute à sa copie le trait `Feature.PRONOMINAL` à `true`. Voir l'exemple 6.2 dans la section sur le formatage de documents (p. 52).

Les exemples 3.1 à 3.5, ci-dessous, illustrent l'utilisation des syntagmes nominaux. L'exemple 3.5 montre aussi comment utiliser les syntagmes adjectivaux et adverbiaux. L'exemple 3.2 montre que quand on spécifie un adverbe comme déterminant, *de* est ajouté juste après celui-ci. L'exemple 3.3 illustre l'utilisation de l'article partitif (*du*). Dans ces exemples, comme précédemment, la fonction `outln(NLGElement)` est utilisée pour ne pas alourdir le code. Sa définition se trouve à l'annexe 2 (p. 64).

Traits :

Trait	Classe du trait	Valeur	Description
`NUMBER`	`Feature`	`NumberAgreement`	Nombre.
`PRONOMINAL`	`Feature`	`Boolean`	Régler à `true` pour que le syntagme nominal soit remplacé par un pronom personnel approprié.
`GENDER`	`LexicalFeature`	`Gender`	Normalement, ce trait reflète automatiquement le genre du nom qui est la tête du syntagme nominal. Cependant, on peut le modifier pour obtenir le nom de genre opposé, s'il a été spécifié par le trait `FrenchLexicalFeature .OPPOSITE_GENDER` du nom.

Exemple 3.1: Adjectifs antéposés et postposés dans un syntagme nominal.

```
NPPhraseSpec snBelEndroit = factory.createNounPhrase("un", "endroit");

// Quelques adjectifs courants sont préposés par défaut.
snBelEndroit.addModifier("beau");
outln(snBelEndroit);

snBelEndroit.clearModifiers();
snBelEndroit.addPostModifier("beau");
outln(snBelEndroit);

// Les autres adjectifs sont postposés par défaut.
snBelEndroit.clearModifiers();
snBelEndroit.addModifier("magnifique");
outln(snBelEndroit);

snBelEndroit.clearModifiers();
snBelEndroit.addPreModifier("magnifique");
outln(snBelEndroit);
```

Sortie :

```
un bel endroit
un endroit beau
un endroit magnifique
un magnifique endroit
```

Exemple 3.2: Adverbes comme déterminants de syntagmes nominaux : la préposition de *est ajoutée.*

```
WordElement assez = lexicon.getWord("assez", LexicalCategory.ADVERB);
NPPhraseSpec snVin = factory.createNounPhrase(assez, "vin");
outln(snVin);

WordElement beaucoup = lexicon.getWord("beaucoup",
   LexicalCategory.ADVERB);
NPPhraseSpec snEleves = factory.createNounPhrase(beaucoup, "élève");
snEleves.setPlural(true);
outln(snEleves);
```

Sortie :

```
assez de vin
beaucoup d'élèves
```

*Exemple 3.3: Article partitif (*du*) dans un syntagme nominal.*

```
NPPhraseSpec snSable = factory.createNounPhrase("du", "sable");
outln(snSable);
NPPhraseSpec snViande = factory.createNounPhrase("du", "viande");
outln(snViande);
NPPhraseSpec snEau = factory.createNounPhrase("du", "eau");
outln(snEau);
```

Sortie :

```
du sable
de la viande
de l'eau
```

Exemple 3.4: Adjectif démonstratif avec particule (séparé de celle-ci par une espace et un trait d'union) dans un syntagme nominal : la particule est apposée au dernier mot du syntagme.

```
NPPhraseSpec snChemise = factory.createNounPhrase("ce -là", "chemise");
snChemise.setPlural(true);
outln(snChemise);

NPPhraseSpec snInstrument = factory.createNounPhrase("ce -ci",
   "instrument");
snInstrument.addModifier("chirurgical");
outln(snInstrument);
```

Sortie :

```
ces chemises-là
cet instrument chirurgical-ci
```

3.2 Syntagme adjectival (`AdjPhraseSpec`)

On spécifie l'adjectif d'un syntagme adjectival soit dans l'appel de la méthode `createAdjectivePhrase`(...) de la *factory*, soit avec la méthode `setAdjective`(...) de Adj`PhraseSpec`.

Pour construire le comparatif analytique (composé de plusieurs mots), quand le comparatif synthétique n'existe pas (voir p. 17), on ajoute un adverbe (*plus*, *moins*, etc.) comme modificateur du syntagme adjectival. Pour construire le superlatif, on procède comme dans l'exemple 4.1 (p. 48), en incluant le syntagme adjectival dans un syntagme nominal où le nom peut subir une ellipse.

3.3 Syntagme adverbial (`AdvPhraseSpec`)

On spécifie l'adverbe d'un syntagme adverbial soit dans l'appel de la méthode `createAdverbPhrase`(...) de la *factory*, soit avec la méthode `setAdverb`(...) de Adv`PhraseSpec`.

Pour construire le comparatif analytique (composé de plusieurs mots), quand le comparatif synthétique n'existe pas (voir p. 20), on ajoute un adverbe *(plus, moins, etc.)* comme modificateur du syntagme adjectival. Pour construire le superlatif, on ajoute d'abord comme modificateur *le*, puis un adverbe comme *plus* ou *moins*.

Exemple 3.5: Syntagmes adverbial, adjectival et nominal.

```
NPPhraseSpec snMaison = factory.createNounPhrase("le", "maison");
AdjPhraseSpec sadjBeau = factory.createAdjectivePhrase("beau");
AdvPhraseSpec sadvTres = factory.createAdverbPhrase("très");
sadjBeau.addModifier(sadvTres);
snMaison.addModifier(sadjBeau);
snMaison.addModifier("spacieux");
snMaison.setPlural(true);
outln(snMaison);
```

Sortie :

```
les très belles maisons spacieuses
```

3.4 Syntagme prépositionnel (`PPPhraseSpec`)

La préposition placée en tête de ce syntagme peut être spécifiée dans l'appel de la méthode `createPrepositionPhrase`(...) de la *factory* ou par la méthode `setPreposition`(...). Le reste du syntagme prépositionnel, habituellement un syntagme nominal, peut être spécifié dans l'appel de la méthode `createPrepositionPhrase`(...) ou par la méthode `setObject`(...).

L'exemple 3.6, qui fait suite à l'exemple 3.5, illustre l'utilisation du syntagme prépositionnel comme complément du nom et l'exemple 3.7, comme complément de l'adjectif. Pour la différence entre modificateur et complément en général dans SimpleNLG, voir la page 22. Dans l'exemple 3.6, on voit que pour un syntagme nominal, sémantiquement, le complément définit avec la tête du syntagme le type d'entité. Les modificateurs, eux, sont des caractéristiques accidentelles de cette entité. En ce qui concerne SimpleNLG, ce qu'il est important de retenir, c'est que les compléments suivent immédiatement le nom, que les postmodificateurs viennent après les compléments et que les syntagmes

prépositionnels utilisés comme modificateurs seront considérés par défaut comme des postmodificateurs.

Exemple 3.6: Syntagmes prépositionnels comme compléments du nom (suite de l'exemple 3.5).

```
PPPhraseSpec complementDuNomCategorie =
   factory.createPrepositionPhrase("de", "campagne");
snMaison.addComplement(complementDuNomCategorie);
PPPhraseSpec complementDuNomMatiere =
   factory.createPrepositionPhrase("en", "pierre");
snMaison.addModifier(complementDuNomMatiere);
outln(snMaison);
```

Sortie :

```
les très belles maisons de campagne spacieuses en pierre
```

Exemple 3.7: Syntagme prépositionnel comme complément de l'adjectif

```
NPPhraseSpec fruits = factory.createNounPhrase("fruit");
fruits.setPlural(true);
PPPhraseSpec complementDeLAdjectif = factory.createPrepositionPhrase("de",
   fruits);
AdjPhraseSpec plein = factory.createAdjectivePhrase("plein");
plein.addComplement(complementDeLAdjectif);
NPPhraseSpec corbeille = factory.createNounPhrase("un", "corbeille");
corbeille.addModifier(plein);
outln(corbeille);
```

Sortie :

```
une corbeille pleine de fruits
```

3.5 Syntagme verbal (`VPPhraseSpec`)

Avant de décrire le syntagme verbal, il est important de souligner que celui-ci n'a pratiquement jamais besoin d'être créé explicitement par l'utilisateur de SimpleNLG ou SimpleNLG-EnFr. Il est en effet possible de spécifier au niveau de la proposition (`SPhraseSpec`) le verbe, les compléments et modificateurs verbaux, de même que les traits relatifs au syntagme verbal énumérés à la page 28. Le syntagme verbal est géré à l'interne par le `SPhraseSpec`. C'est pourquoi il n'y a pas d'exemple sur l'utilisation du `VPPhraseSpec`; il faut se référer aux exemples 3.8 à 3.10, qui portent sur la proposition. Donc, on peut régler sur la proposition tout ce qui a besoin d'être réglé dans le syntagme verbal. Cependant, si cela s'avère utile, on peut tout de même créer un syntagme verbal seul et le manipuler.

Le verbe est spécifié soit dans l'appel de la méthode `createVerbPhrase`(...) de la *factory*, soit avec la

méthode `setVerb`(...). Les compléments d'objet direct et indirect peuvent être spécifiés à l'aide des méthodes `setObject`(...) et `setIndirectObject`(...) respectivement. Un complément d'objet indirect peut être spécifié de deux manières : par un syntagme prépositionnel ou par un syntagme nominal. Dans ce dernier cas, la préposition par défaut *à* lui sera attribuée.

À l'intérieur du groupe des compléments, trois sous-groupes sont formés : les compléments d'objet direct, les compléments d'objet indirect et les autres compléments. Ces trois sous-groupes sont placés préférablement dans cet ordre s'ils comptent le même nombre de mots; sinon, ils sont placés en ordre croissant de nombre de mots.

Les pronoms compléments verbaux clitiques forment un cas particulier. Ils sont placés selon les règles usuelles, recensées dans Grevisse (1993 : §657, p. 1044-1045). Ils sont la plupart du temps placés juste avant le premier élément verbal conjugué. À l'impératif et en l'absence de négation, ils sont placés après ce dernier et lui sont rattachés par un trait d'union. Quelques autres règles s'appliquent dans des cas plus spécifiques.

Quand on ajoute un modificateur à un syntagme verbal avec la méthode add`Modifier(...)`, il est considéré par défaut comme un postmodificateur. Les prémodificateurs verbaux sont placés immédiatement avant le verbe principal en présence d'un auxiliaire et immédiatement après sinon. Viennent ensuite les compléments suivis des postmodificateurs. Un complément adverbial, par exemple, peut donc être placé de trois façons (en gras ci-dessous) à l'intérieur du syntagme verbal : prémodificateur, complément ou postmodificateur.

Sans auxiliaire verbal, nous avons donc l'ordre suivant :

verbe→**prémod.**→compléments (OD↔OI↔**autres**)→**postmod.**

ou, avec auxiliaire(s) :

auxiliaire(s)→**prémod.**→verbe principal→compléments (OD↔OI↔**autres**)→**postmod.**

OD : objets directs, OI : objets indirects
prémod. : prémodificateurs, postmod. : postmodificateurs
→: Ordre fixe, tel qu'indiqué de gauche à droite.
↔: Ordre croissant du nombre de mots de chaque groupe. L'ordre indiqué est l'ordre préférentiel si le nombre de mots est le même dans les trois groupes de compléments.

Les traits permettant de spécifier les temps verbaux en anglais ont été conservés tels quels en français pour assurer la compatibilité avec la partie anglaise. Toutefois, une idée exprimée en anglais par une combinaison donnée de traits verbaux pourrait être mieux traduite, dans un contexte donné, par une combinaison différente de traits verbaux en français. Par exemple, « *I'm walking in the street.* » (progressif) pourrait mieux se traduire, dans certains contextes, par « *Je marche dans la rue. »* (non progressif). Il appartient à l'utilisateur de SimpleNLG-EnFr de tenir compte de ces différences s'il compte générer du texte dans les deux langues.

Le passé récent (avec *venir de*) n'est pas disponible directement dans SimpleNLG-EnFr, mais on peut le construire en créant une syntagme verbal avec venir comme verbe et en lui ajoutant comme complément soit une proposition infinitive avec *de* comme complémenteur (voir la section sur la proposition), soit un syntagme prépositionnel avec *de* comme préposition et un verbe ou syntagme verbal à l'infinitif comme objet de ce syntagme prépositionnel.

Pour le futur proche, il suffit de spécifier *aller* comme « modal » (`Feature.MODAL`).

Le temps utilisé au passé non progressif et non parfait est le passé composé. Le passé simple n'est pas disponible dans la partie française de SimpleNLG-EnFr. L'imparfait est utilisé pour le passé non parfait progressif. Dans les autres cas, *être en train de* est ajouté comme auxiliaire pour traduire l'aspect progressif.

Traits :

Trait	Classe du trait	Valeur	Description
`TENSE`	`Feature`	`Tense`	Temps (passé, présent, futur, conditionnel).
`FORM`	`Feature`	`Form`	Forme (participe présent, participe passé, indicatif (`NORMAL`), gérondif, impératif, subjonctif).
`PERFECT`	`Feature`	`Boolean`	Régler à `true` pour mettre le verbe à un temps « parfait », c'est-à-dire plus-que-parfait (passé), passé composé (présent) ou futur antérieur (futur).
`PROGRESSIVE`	`Feature`	`Boolean`	Régler à `true` pour mettre le verbe à une forme « progressive ». (voir plus haut)
`MODAL`	`Feature`	`String`	S'il y a lieu, spécifier le verbe auxiliaire modal à utiliser à l'aide de ce trait.
`NEGATED`	`Feature`	`Boolean`	Régler à `true` pour mettre le syntagme verbal à la négative. (*ne...pas*) Le trait `NE_ONLY_NEGATION` (voir p. 14) empêche la réalisation de l'auxiliaire de négation.
`PASSIVE`	`Feature`	`Boolean`	Régler à `true` pour mettre le syntagme verbal au passif.
`NEGATION_AUXILIARY`	`FrenchFeature`	`String` `NLGElement`	Permet de spécifier un auxiliaire de négation : *pas*, *plus*, *point*, *guère*, *aucunement*, etc.

Si le trait `NEGATION_AUXILIARY` est laissé à `null`, *pas* est utilisé comme auxiliaire de négation par défaut. L'auxiliaire de négation *plus* est le seul qui soit conservé dans une proposition mise à la négative avec le trait `NEGATED` et ayant comme sujet ou objet direct ou indirect un mot entraînant la négation avec *ne* seulement.[5]

Le tableau 1 illustre l'effet des combinaisons des traits anglais `TENSE` (temps), `PROGRESSIVE` (aspect progressif) et `PERFECT` (aspect parfait) sur le verbe français *manger* et sur le verbe anglais *to eat*, à la 3e personne du singulier. Le temps français est noté entre parenthèses. Notez bien qu'il ne s'agit pas de traductions fixes des temps verbaux anglais correspondants, mais d'«équivalents» pouvant être utiles dans la réalisation de textes en français.

Dans SimpleNLG-EnFr 1.1, le conditionnel et le subjonctif ont été ajoutés à la conjugaison française.

5 Voir la description du trait `FrenchLexicalFeature.NE_ONLY_NEGATION` à la page 14.

Pour y avoir accès, on utilise la nouvelle valeur CONDITIONAL du trait TENSE et la nouvelle valeur SUBJUNCTIVE du trait FORM, respectivement. Les tableaux 2 et 3 montrent le résultat de ces ajouts. Pour préserver l'uniformité du traitement, en anglais, le conditionnel est réalisé en ajoutant le modal *could*. L'implémentation du subjonctif anglais n'ayant pas été jugée prioritaire, il est quant à lui traité de manière identique à l'indicatif.

Tableau 1: Conjugaison de l'indicatif (trait FORM : NORMAL)

Traits anglais d'aspect		Trait TENSE		
PROGRESSIVE	PERFECT	PAST	PRESENT	FUTURE
FAUX	FAUX	il a mangé (passé composé) *he ate*	il mange (présent) *he eats*	il mangera (futur simple) *he will eat*
	VRAI	il avait mangé (plus-que-parfait) *he had eaten*	il a mangé (passé composé) *he has eaten*	il aura mangé (futur antérieur) *he will have eaten*
VRAI	FAUX	il mangeait (imparfait) *he was eating*	il est en train de manger (aux. e*n train de* présent) *he is eating*	il sera en train de manger (aux. e*n train de* futur simple) *he will be eating*
	VRAI	il avait été en train de manger (aux. e*n train de* plus-que-parfait) *he had been eating*	il a été en train de manger (aux. e*n train de* passé composé) *he has been eating*	il aura été en train de manger (aux. e*n train de* futur antérieur) *he will have been eating*

Tableau 2: Conjugaison du conditionnel (trait FORM : NORMAL)

Traits anglais d'aspect		Trait TENSE
PROGRESSIVE	PERFECT	CONDITIONAL
FAUX	FAUX	il mangerait (conditionnel présent) *he could eat*
	VRAI	il aurait mangé (conditionnel passé) *he could have eaten*
VRAI	FAUX	il serait en train de manger (aux. e*n train de* conditionnel présent) *he could be eating*
	VRAI	il aurait été en train de manger (aux. e*n train de* conditionnel passé) *he could have been eating*

Tableau 3: Conjugaison du subjonctif (trait FORM : SUBJUNCTIVE)

Traits anglais d'aspect		Trait TENSE	
PROGRESSIVE	PERFECT	PAST	PRESENT, FUTURE, CONDITIONAL
FAUX	FAUX	qu'il ait mangé (passé)	qu'il mange (présent)
	VRAI	qu'il ait eu mangé (passé surcomposé)	qu'il ait mangé (passé)
VRAI	FAUX	qu'il ait été en train de manger (aux. e*n train de* passé)	qu'il soit en train de manger (aux. e*n train de* présent)
	VRAI	qu'il ait eu été en train de manger (aux. e*n train de* passé surcomposé)	qu'il ait été en train de manger (aux. e*n train de* passé)

3.6 Proposition (SphraseSpec)

3.6.1 Considérations générales

Le sujet, le verbe ou le syntagme verbal et l'objet direct peuvent être spécifiés soit dans l'appel de la méthode `createClause`(...) de la *factory*, soit avec les méthodes `setSubject`(...), `setVerb`(...), `setVerbPhrase`(...) et `setObject`(...) respectivement. S'il y a un attribut du sujet au lieu d'un objet direct, il est tout de même spécifié avec `setObject`(...). Le complément d'objet indirect peut être spécifié à l'aide de la méthode `setIndirectObject`(...). Il peut être spécifié de deux manières : par un syntagme prépositionnel ou par un syntagme nominal. Dans ce dernier cas, la préposition par défaut *à* lui sera attribuée.

Il est possible de spécifier au niveau de la proposition le verbe, les compléments et modificateurs verbaux, de même que les traits relatifs au syntagme verbal énumérés à la page 28. C'est même préférable dans le cas des traits. Le verbe, les traits verbaux, les compléments et les pré- et postmodificateurs de la proposition sont transmis automatiquement à son syntagme verbal.

En plus des pré- et postmodificateurs, les propositions peuvent aussi avoir des modificateurs placés en tête de phrase (`addFrontModifier(...)`). Ces modificateurs sont placés juste après le complément spécifié par le trait `Feature.CUE_PHRASE`, s'il y en a un. Ils sont par défaut suivi d'une virgule. Pour qu'ils ne soient pas chacun suivis d'une virgule, il suffit de régler le trait `FrenchLexicalFeature.NO_COMMA` de chacun de ces éléments à `true`. Si la proposition est de forme infinitive, ces modificateurs sont placés après les postmodificateurs, sans virgule.

L'élément spécifié par le trait `CUE_PHRASE` (ci-dessous) est par défaut suivi d'une virgule. Pour qu'il ne soit pas suivi d'une virgule, il suffit de régler le trait `FrenchLexicalFeature.NO_COMMA` de cet élément à `true`. Cet élément n'est pas réalisé en français si la proposition est de forme infinitive.

Les exemples 3.8 à 3.10 illustrent l'utilisation de la proposition, l'attribution de ses compléments et le réglage des temps verbaux. L'exemple 3.10 comporte des propositions subordonnées. L'ajout de la conjonction de subordination *si* entraîne la transformation du futur simple en présent, du futur antérieur en passé composé, du conditionnel présent en imparfait et du conditionnel passé en plus-que-parfait (Grevisse 1993 : §1097, p. 1687-1688). L'exemple 3.10 illustre le cas du conditionnel présent.

Traits :

Trait	Classe du trait	Valeur	Description
`CUE_PHRASE`	`Feature`	`NLGElement`	Spécifie un élément qui doit être placé en tête de phrase.
`COMPLEMENTISER`	`Feature`	`String` `NLGElement`	Spécifie une conjonction de subordination (« *complementiser* » ou complémenteur) autre que celle par défaut, c'est-à-dire *que* pour le français et *that* pour l'anglais.
`SUPRESSED_COMPLEMENTISER`	`Feature`	`Boolean`	Régler à `true` pour empêcher la réalisation du complémenteur de la proposition. Il n'est la plupart du temps pas nécessaire que l'utilisateur s'occupe de ce trait.
`INTERROGATIVE_TYPE`	`Feature`	`InterrogativeType`	Détermine le type d'interrogation pour cette proposition. Une valeur `null` indique une proposition déclarative.
`RELATIVE_PHRASE`	`FrenchFeature`	`PhraseElement`	Si non `null`, indique quel syntagme doit être remplacé par un pronom relatif pour former une proposition relative.

*Exemple 3.8: Conjugaison à l'imparfait (*tomber*), au présent (*rire*) et au futur simple (*aboyer*) de l'indicatif avec différents pronoms et modificateurs adverbiaux.*

```
SPhraseSpec proposition = factory.createClause("on", "tomber");
proposition.setFeature(Feature.TENSE, Tense.PAST);
proposition.setFeature(Feature.PROGRESSIVE, true);
proposition.addModifier("souvent");
outln(proposition);

proposition = factory.createClause("vous", "rire");
proposition.addModifier("beaucoup");
outln(proposition);

proposition = factory.createClause("ils", "aboyer");
proposition.setFeature(Feature.TENSE, Tense.FUTURE);
proposition.addModifier("fort");
outln(proposition);
```

Sortie :

```
on tombait souvent
vous riez beaucoup
ils aboieront fort
```

Exemple 3.9: Le verbe vendre *est conjugué au passé composé de l'indicatif et au présent de l'impératif. On a ajouté un prémodificateur (*rapidement*), un* frontmodifier *(*demain*), ou un modificateur (*avant demain*). Ce dernier est placé par défaut en tant que postmodificateur. Un syntagme nominal est remplacé par un pronom (*les maisons→les*). L'objet indirect entré sous la forme* ils *est correctement réalisé dans la forme* leur. *Les pronoms clitiques compléments du verbe (*les, leur*) sont correctement placés, de même que les adverbes de négation (*ne ... pas*).*

```
proposition = factory.createClause("elle", "vendre");
proposition.addPreModifier("rapidement");
proposition.setFeature(Feature.TENSE, Tense.PAST);
proposition.setObject(snMaison);
snMaison.setPlural(true);
outln(proposition);

proposition.setSubject("tu");
snMaison.setFeature(Feature.PRONOMINAL, true);
outln(proposition);

proposition.setFeature(Feature.FORM, Form.IMPERATIVE);
proposition.setFeature(Feature.TENSE, Tense.PRESENT);
proposition.setIndirectObject("ils");
proposition.clearModifiers();
proposition.addFrontModifier("demain");
outln(proposition);

proposition.setFeature(Feature.NEGATED, true);
proposition.clearModifiers();
PPPhraseSpec avantDemain =
     factory.createPrepositionPhrase("avant", "demain");
proposition.addModifier(avantDemain);
outln(proposition);
```

Sortie :

```
elle a rapidement vendu les maisons
tu les as rapidement vendues
demain, vends-les-leur
ne les leur vends pas avant demain
```

Exemple 3.10: Plus-que-parfait de l'indicatif (verbe aller*, avec l'auxiliaire* être*), propositions subordonnées complétives et changement de la conjonction de subordination* que *pour* si*. Changement de temps entraîné par* si *: le conditionnel présent devient l'imparfait.*

```
proposition = factory.createClause("elles", "aller");
proposition.setFeature(Feature.TENSE, Tense.PAST);
proposition.setFeature(Feature.PERFECT, true);
NPPhraseSpec snLeParc = factory.createNounPhrase("le", "parc");
PPPhraseSpec spAuParc = factory.createPrepositionPhrase("à", snLeParc);
proposition.setComplement(spAuParc);
outln(proposition);

SPhraseSpec proposition2 = factory.createClause("vous", "dire",
   proposition);
outln(proposition2);

SPhraseSpec proposition3 = factory.createClause("il", "être", "riche");
proposition3.setFeature(Feature.TENSE, Tense.CONDITIONAL);
proposition3.setFeature(Feature.COMPLEMENTISER, "si");
snMaison = factory.createNounPhrase("un", "maison");
SPhraseSpec proposition4 =
     factory.createClause("il", "acheter", snMaison);
proposition4.setFeature(Feature.TENSE, Tense.CONDITIONAL);
proposition4.addFrontModifier(proposition3);
outln(proposition4);
```

Sortie :

```
elles étaient allées au parc
vous dites qu'elles étaient allées au parc
s'il était riche, il achèterait une maison
```

Exemple 3.11: Le verbe pouvoir *est mis en tant que verbe modal. Le pronom indéfini négatif* personne *provoque la négation avec* ne *seulement. L'auxiliaire de négation est ensuite changé pour* plus*.*

```
SPhraseSpec proposition5 = factory.createClause("personne", "marcher");
proposition5.setFeature(Feature.MODAL, "pouvoir");
outln(proposition5);

proposition5.setSubject("moi");
proposition5.setFeature(Feature.NEGATED, true);
outln(proposition5);

proposition5.setFeature(FrenchFeature.NEGATION_AUXILIARY, "plus");
outln(proposition5);
```

Sortie :

```
personne ne peut marcher
je ne peux pas marcher
je ne peux plus marcher
```

3.6.2 Proposition relative

Dans la version 1.1 de SimpleNLG-EnFr, la possibilité de demander la formation automatique d'une proposition relative en français a été ajoutée. Il suffit pour cela d'attribuer comme valeur au trait `FrenchFeature.RELATIVE_PHRASE` le syntagme que l'on veut remplacer par un pronom relatif. Généralement, il s'agit d'un syntagme nominal ou prépositionnel. Dans l'exemple 3.12, les propositions relatives sont présentées de manière isolée, sans être rattachées à un syntagme nominal. L'exemple 3.14, p. 37, montre ces mêmes propositions relatives en tant que modificateurs de syntagmes nominaux.

Exemple 3.12: Formation de proposition relative par désignation du syntagme relativisé.

```
NPPhraseSpec snAgent = factory.createNounPhrase("le", "agent");
snAgent.addModifier("immobilier");
NPPhraseSpec snMaison = factory.createNounPhrase("le", "maison");
snMaison.setPlural(true);
NPPhraseSpec snCliente = factory.createNounPhrase("le", "cliente");
NPPhraseSpec snBureau = factory.createNounPhrase("son", "bureau");
PPPhraseSpec   spDansBureau   =   factory.createPrepositionPhrase("dans",
snBureau);
SPhraseSpec   proposition6   =   factory.createClause(snAgent,   "vendre",
snMaison);
proposition6.setIndirectObject(snCliente);
proposition6.addComplement(spDansBureau);
proposition6.setFeature(Feature.TENSE, Tense.PAST);
outln(proposition6);

proposition6.setFeature(FrenchFeature.RELATIVE_PHRASE, snAgent);
outln(proposition6);
proposition6.setFeature(FrenchFeature.RELATIVE_PHRASE, snMaison);
outln(proposition6);
proposition6.setFeature(FrenchFeature.RELATIVE_PHRASE, snCliente);
outln(proposition6);
proposition6.setFeature(FrenchFeature.RELATIVE_PHRASE, spDansBureau);
outln(proposition6);
```

Sortie :

```
l'agent immobilier a vendu les maisons à la cliente dans son bureau
qui a vendu les maisons à la cliente dans son bureau
que l'agent immobilier a vendu à la cliente dans son bureau
auquel l'agent immobilier a vendu les maisons dans son bureau
dans lequel l'agent immobilier a vendu les maisons à la cliente
```

Il peut arriver qu'on ne connaissent pas l'identité du syntagme à relativiser, ou qu'il n'existe pas réellement, mais qu'on connaisse sa fonction dans la proposition. Dans ce cas, on peut créer un syntagme factice et l'utiliser pour spécifier le type de syntagme à relativiser. Il est possible alors de ne spécifier que la fonction (trait `InternalFeature.DISCOURSE_FUNCTION`) ou la préposition du syntagme, comme dans l'exemple 3.13.

Exemple 3.13: Formation de proposition relative par utilisation d'un syntagme factice.

```
NPPhraseSpec snFactice = factory.createNounPhrase();
snFactice.setFeature(InternalFeature.DISCOURSE_FUNCTION,
     DiscourseFunction.SUBJECT);
proposition6.setFeature(FrenchFeature.RELATIVE_PHRASE, snFactice);
outln(proposition6);

PPPhraseSpec spFactice = factory.createPrepositionPhrase("lors de");
proposition6.setFeature(FrenchFeature.RELATIVE_PHRASE, spFactice);
outln(proposition6);
```

Sortie :

```
qui a vendu les maisons à la cliente dans son bureau
lors duquel l'agent immobilier a vendu les maisons à la cliente dans son
bureau
```

Si le syntagme désigné a la fonction de sujet, d'objet direct ou d'objet indirect, aucun syntagme de même fonction ne sera réalisé dans la proposition relative.

Le pronom relatif est choisi selon la fonction du syntagme relativisé et sa préposition, le cas échéant. D'autre part, il a comme antécédent la tête du syntagme nominal que la proposition relative modifie; il en prend donc le genre, le nombre, la personne et la propriété d'être un nom propre (trait `LexicalFeature.PROPER`). Cela peut modifier la conjugaison du verbe de la proposition relative, en particulier si l'objet direct est relativisé en présence d'un participe passé avec l'auxiliaire *avoir*. Dans ce cas, l'objet direct étant placé devant le participe passé, ce dernier s'accorde avec la tête du syntagme nominal que la proposition relative modifie. On peut observer ce phénomène dans la 2^{e} réalisation de l'exemple 3.14.

Le pronom relatif par défaut en présence d'une préposition est *lequel*. Quand l'antécédent est de genre neutre (ce qui arrive rarement), il est remplacé par le pronom relatif *quoi*, comme dans *ce à quoi*. Dans le même contexte, le pronom relatif *qui* est utilisé lorsqu'on est sûr que l'antécédent est une personne (physique ou morale). En pratique, comme SimpleNLG-EnFr n'a pas accès à des indications sémantiques, *qui* est utilisé seulement en présence d'un antécédent de 1ère ou 2^{e} personne ou d'un nom propre marqué comme tel. Les 3^{e}, 4^{e} et 5^{e} réalisations de l'exemple 3.14 illustrent ces principes.

Exemple 3.14: Proposition relative comme modificateur d'un syntagme nominal.

```
NPPhraseSpec snPrincipal = factory.createNounPhrase("le", "personne");
snPrincipal.addModifier(proposition6);
proposition6.setFeature(FrenchFeature.RELATIVE_PHRASE, snAgent);
outln(snPrincipal);

snPrincipal.setNoun("propriété");
proposition6.setFeature(FrenchFeature.RELATIVE_PHRASE, snMaison);
outln(snPrincipal);

snPrincipal.setNoun("personne");
proposition6.setFeature(FrenchFeature.RELATIVE_PHRASE, snCliente);
outln(snPrincipal);
snPrincipal.setSpecifier(null);
snPrincipal.setPronoun("vous");
outln(snPrincipal);
WordElement JeanPierre = lexicon.lookupWord("Jean-Pierre",
     LexicalCategory.NOUN);
JeanPierre.setFeature(LexicalFeature.PROPER, true);
snPrincipal.setNoun(JeanPierre);
outln(snPrincipal);

snPrincipal.setSpecifier("le");
snPrincipal.setNoun("pièce");
proposition6.setFeature(FrenchFeature.RELATIVE_PHRASE, spDansBureau);
outln(snPrincipal);
```

Sortie :

```
la personne qui a vendu les maisons à la cliente dans son bureau
la propriété que l'agent immobilier a vendue à la cliente dans son bureau
la personne à laquelle l'agent immobilier a vendu les maisons dans son
bureau
vous à qui l'agent immobilier a vendu les maisons dans son bureau
Jean-Pierre à qui l'agent immobilier a vendu les maisons dans son bureau
la pièce dans laquelle l'agent immobilier a vendu les maisons à la cliente
```

Le cas du pronom relatif *dont* est particulier. Il remplace un syntagme prépositionnel avec la préposition *de* et ce syntagme peut être un complément du verbe, mais peut aussi être le complément d'un nom. L'exemple 3.15 illustre ce dernier cas. La 1ère réalisation montre la proposition de base, la 2e la proposition relative isolée, dont le complément du nom sujet a été relativisé, et la 3e montre la proposition relative comme modificateur d'un syntagme nominal.

Exemple 3.15: Proposition relative avec le pronom relatif dont.

```
NPPhraseSpec snMinistre = factory.createNounPhrase("le", "ministre");
PPPhraseSpec spDuMinistre = factory.createPrepositionPhrase("de",
     snMinistre);
snMaison.addComplement(spDuMinistre);
SPhraseSpec proposition7 = factory.createClause(null, "vendre", snMaison);
proposition7.setFeature(Feature.PASSIVE, true);
proposition7.setFeature(Feature.TENSE, Tense.PAST);
outln(proposition7);

proposition7.setFeature(FrenchFeature.RELATIVE_PHRASE, spDuMinistre);
outln(proposition7);

snPrincipal = factory.createNounPhrase("le", "homme");
snPrincipal.addModifier("politique");
snPrincipal.addModifier("célèbre");
snPrincipal.addModifier(proposition7);
outln(snPrincipal);
```

Sortie :

```
les maisons du ministre ont été vendues
dont les maisons ont été vendues
l'homme politique célèbre dont les maisons ont été vendues
```

3.6.3 Proposition interrogative

Dans la version 1.1 de SimpleNLG-EnFr, la fonctionnalité permettant de demander la formation d'une proposition interrogative en anglais a été étendue au français. Les types de question qu'il est possible d'obtenir, qui ne couvrent que les principales possibilités, sont les mêmes que dans la version originale anglaise. Les exemples 3.16, 3.17 et 3.18 montrent comment construire des propositions interrogatives.

Dans SimpleNLG-EnFr, la formulation de questions avec l'introducteur interrogatif *est-ce que* a été choisie, plutôt que l'inversion du pronom sujet, parce que la première appartient davantage à la langue courante que la deuxième. De fait, l'emploi de *est-ce que* est grammaticalement beaucoup plus régulier. Dans certains cas, il est même grammaticalement impossible d'utiliser l'inversion du pronom sujet et l'emploi de l'introducteur *est-ce que* devient obligatoire (ex. : **Perds-je la tête?* vs *Est-ce que je perds la tête?*). Pour plus d'informations, voir Grevisse (1993 : §386-389, p. 639-653).

En français, une interrogation globale (dont on attend une réponse par *oui* ou *non*) est exprimée différemment selon qu'il s'agisse d'une interrogation directe ou indirecte. Dans le cas d'une interrogation directe, c'est-à-dire quand la proposition interrogative apparaît comme proposition principale de la phrase, l'introducteur interrogatif *est-ce que* est employé. Dans le cas d'une interrogation indirecte, c'est-à-dire quand la proposition interrogative apparaît comme proposition subordonnée complétive, l'introducteur interrogatif *si* est plutôt employé. Il est à noter que ce *si* interrogatif ne provoque pas de changement dans le temps du verbe de la proposition, contrairement au *si* de condition (Grevisse 1993 : §1098, p. 1689). Voir les explications pour le *si* de condition p. 30 et 3e réalisation de l'exemple 3.10, p. 33. L'exemple 3.18, p. 44, illustre l'utilisation des proposition interrogatives indirectes.

Dans les exemples 3.16, 3.17 et 3.18 sur les propositions interrogatives, on utilise la méthode `realiseSentence(...)` pour que la majuscule de début de phrase et la ponctuation de fin de phrase soient réalisées, ceci dans le but de faciliter la compréhension des phrases interrogatives. Pour plus de détails sur la réalisation de la phrases complètes, voir la section 6 Formatage du document, p. 51.

Exemple 3.16: Propositions interrogatives globales et partielles, avec qui *sujet ou objet, à l'actif et au passif.*

```
PhraseElement proposition8 = factory.createClause(
     factory.createNounPhrase("le", "femme"),
     "embrasser",
     factory.createNounPhrase("le", "homme"));
System.out.println( realiser.realiseSentence(proposition8) );

proposition8.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.YES_NO);
System.out.println( realiser.realiseSentence(proposition8) );

proposition8.setFeature(Feature.PASSIVE, true);
System.out.println( realiser.realiseSentence(proposition8) );

proposition8.setFeature(Feature.PASSIVE, false);
proposition8.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.WHO_SUBJECT);
System.out.println( realiser.realiseSentence(proposition8) );

proposition8.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.WHO_OBJECT);
System.out.println( realiser.realiseSentence(proposition8) );

proposition8.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.WHO_SUBJECT);
proposition8.setFeature(Feature.PASSIVE, true);
System.out.println( realiser.realiseSentence(proposition8) );

proposition8.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.WHO_OBJECT);
System.out.println( realiser.realiseSentence(proposition8) );
```

Sortie :

```
La femme embrasse l'homme.
Est-ce que la femme embrasse l'homme?
Est-ce que l'homme est embrassé par la femme?
Qui est-ce qui embrasse l'homme?
Qui est-ce que la femme embrasse?
Par qui est-ce que l'homme est embrassé?
Qui est-ce qui est embrassé par la femme?
```

Exemple 3.17: Autres types de propositions interrogatives partielles.

```
NPPhraseSpec snAgent = factory.createNounPhrase("le", "agent");
snAgent.addModifier("immobilier");
NPPhraseSpec snMaison = factory.createNounPhrase("un", "maison");
NPPhraseSpec snCliente = factory.createNounPhrase("le", "cliente");
SPhraseSpec proposition9 =
     factory.createClause(snAgent, "vendre", snMaison);
proposition9.setIndirectObject(snCliente);
proposition9.setFeature(Feature.TENSE, Tense.PAST);
System.out.println( realiser.realiseSentence(proposition9) );

proposition9.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.WHO_INDIRECT_OBJECT);
System.out.println( realiser.realiseSentence(proposition9) );

proposition9.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.WHAT_OBJECT);
System.out.println( realiser.realiseSentence(proposition9) );

proposition9.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.HOW);
System.out.println( realiser.realiseSentence(proposition9) );

proposition9.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.WHERE);
System.out.println( realiser.realiseSentence(proposition9) );

proposition9.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.WHY);
System.out.println( realiser.realiseSentence(proposition9) );

proposition9.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.HOW_MANY);
System.out.println( realiser.realiseSentence(proposition9) );

proposition9.setFeature(Feature.PASSIVE, true);
System.out.println( realiser.realiseSentence(proposition9) );
```

Sortie :

```
L'agent immobilier a vendu une maison à la cliente.
À qui est-ce que l'agent immobilier a vendu une maison?
Qu'est-ce que l'agent immobilier a vendu à la cliente?
Comment est-ce que l'agent immobilier a vendu une maison à la cliente?
Où est-ce que l'agent immobilier a vendu une maison à la cliente?
Pourquoi est-ce que l'agent immobilier a vendu une maison à la cliente?
Combien d'agents immobiliers ont vendu une maison à la cliente?
Combien de maisons ont été vendues à la cliente par l'agent immobilier?
```

Exemple 3.18: Propositions interrogatives directes et indirectes. Est-ce que *est remplacé par* si *dans une proposition interrogative globale indirecte. Le temps du verbe n'est pas affecté par ce* si.

```
proposition9.setFeature(Feature.PASSIVE, false);
proposition9.setFeature(Feature.TENSE, Tense.CONDITIONAL);
proposition9.setFeature(Feature.PERFECT, true);
proposition9.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.YES_NO);
System.out.println( realiser.realiseSentence(proposition9) );

SPhraseSpec propositionPrincipale =
     factory.createClause("vous", "dire", proposition9);
propositionPrincipale.setIndirectObject("nous");
propositionPrincipale.setFeature(Feature.FORM, Form.IMPERATIVE);
System.out.println( realiser.realiseSentence(propositionPrincipale) );

proposition9.setFeature(Feature.INTERROGATIVE_TYPE,
     InterrogativeType.WHO_INDIRECT_OBJECT);
System.out.println( realiser.realiseSentence(proposition9) );

System.out.println( realiser.realiseSentence(propositionPrincipale) );
```

Sortie :

```
Est-ce que l'agent immobilier aurait vendu une maison à la cliente?
Dites-nous si l'agent immobilier aurait vendu une maison à la cliente.
À qui est-ce que l'agent immobilier aurait vendu une maison?
Dites-nous à qui est-ce que l'agent immobilier aurait vendu une maison.
```

3.7 Syntagme coordonné (`CoordinatedPhraseElement`)

La conjonction de coordination utilisée par ce syntagme peut être spécifiée par la méthode `setConjunction`(...). Par défaut, c'est la conjonction *et* qui sera utilisée. Les éléments coordonnés, qui peuvent être de n'importe quel type, peuvent être ajoutés avec la méthode add`Coordinate`(...). On peut aussi en spécifier deux dès le départ dans l'appel de la méthode `createCoordinatedPhrase`(...) de la *factory*.

Traits :

Trait	Classe du trait	Valeur	Description
`AGGREGATE_AUXILIARY`	`Feature`	`Boolean`	Régler à `true` pour mettre en commun (agréger) les auxiliaires de syntagmes verbaux coordonnés.

L'exemple 3.19 illustre l'utilisation du syntagme coordonné et la manière dont il influe sur le reste de la proposition.

*Exemple 3.19: Coordination de syntagmes nominaux sujets (*Marie*,* Julie*) et changement de la conjonction de coordination* et *pour* ou*.*

```
SPhraseSpec proposition = factory.createClause();
proposition.setVerb("devenir");
NPPhraseSpec snMarie = factory.createNounPhrase("Marie");
snMarie.setFeature(LexicalFeature.GENDER, Gender.FEMININE);
NPPhraseSpec snJulie = factory.createNounPhrase("Julie");
snJulie.setFeature(LexicalFeature.GENDER, Gender.FEMININE);
CoordinatedPhraseElement coord = factory.createCoordinatedPhrase();
coord.addCoordinate(snMarie);
coord.addCoordinate(snJulie);
proposition.setSubject(coord);
proposition.setObject("gentil");
proposition.setFeature(Feature.TENSE, Tense.FUTURE);
proposition.setFeature(Feature.PERFECT, true);
outln(proposition);

coord.setConjunction("ou");
outln(proposition);
```

Sortie :

```
Marie et Julie seront devenues gentilles
Marie ou Julie sera devenue gentille
```

3.8 « Texte en boîte » (« canned text », `StringElement`)

Les groupes de mots dont on n'a pas besoin de connaître la structure peuvent être entrés en bloc sous la forme d'un `StringElement`. Cette classe dérivée de `NLGElement` est un conteneur pour une chaîne de caractères arbitraire. Il est possible de spécifier des traits si nécessaire.

L'exemple 3.20 illustre l'utilisation des `StringElement` et la façon dont on peut les combiner librement avec les autres types d'éléments. Il montre aussi une autre méthode pour sélectionner un pronom, de même qu'un aperçu de la façon dont les compléments pronominaux clitiques sont gérés.

Exemple 3.20: On peut insérer du « texte en boîte » à n'importe quel endroit de la structure syntaxique.

```
StringElement peche = factory.createStringElement("à la pêche au requin");
StringElement pierre = factory.createStringElement("Pierre part à la
   chasse");

Map<String,Object> features = new HashMap<String,Object>();
features.put(Feature.PERSON, Person.FIRST);
features.put(Feature.NUMBER, NumberAgreement.SINGULAR);
WordElement pronom = lexicon.getWord(LexicalCategory.PRONOUN, features);
SPhraseSpec proposition2 = factory.createClause(pronom, "aller", "se");
proposition2.addComplement("en");
proposition2.addComplement(peche);
proposition2.addPreModifier("ensuite");

CoordinatedPhraseElement coord = factory.createCoordinatedPhrase();
coord.addCoordinate(pierre);
coord.addCoordinate(proposition2);
outln(proposition);
```

Sortie :

```
Pierre part à la chasse et je m'en vais ensuite à la pêche au requin
```

4 Ellipse d'un mot ou d'un syntagme

Le trait suivant peut s'appliquer à tous les `NLGElement`, sauf les `WordElement` :

Trait	Classe du trait	Valeur	Description
`ELIDED`	`Feature`	`Boolean`	Permet de spécifier qu'un élément subit une ellipse et ne doit donc pas être réalisé.

En général, pour modifier les traits d'une instance particulière d'une unité lexicale, par opposition aux valeurs des traits que cette unité lexicale possède dans le lexique, il faut d'abord créer un nouveau `InflectedWordElement` en donnant le `WordElement` souhaité en argument au constructeur. (Il n'est pas possible d'utiliser la *factory* pour ce faire.) L'exemple 4.1 montre comment le faire avec le trait `ELIDED`. Le procédé est le même si on désire effectuer la flexion de mots isolés (voir p. 13). Notez par contre que pour insérer un `InflectedWordElement` dans un syntagme nominal, on doit avoir recours à la méthode `setHead`(...) au lieu de `setNoun`(...).

Exemple 4.1: Utilisation du trait `ELIDED` pour l'ellipse d'une unité lexicale.

```
WordElement qualite = lexicon.getWord("qualité");
InflectedWordElement qualiteElided = new InflectedWordElement(qualite);
qualiteElided.setFeature(Feature.ELIDED, true);

NPPhraseSpec toutQualite = factory.createNounPhrase("tout", qualite);
toutQualite.setPlural(true);
PPPhraseSpec deToutesLesQualites =
     factory.createPrepositionPhrase("de", toutQualite);

NPPhraseSpec laPlusImportante = factory.createNounPhrase();
laPlusImportante.setHead(qualiteElided);
laPlusImportante.setSpecifier("le");
AdjPhraseSpec plusImportant = factory.createAdjectivePhrase("important");
plusImportant.addModifier("plus");
laPlusImportante.addPreModifier(plusImportant);

NPPhraseSpec leCourage = factory.createNounPhrase("le", "courage");

SPhraseSpec proposition =
     factory.createClause(leCourage, "être", laPlusImportante);
proposition.addFrontModifier(deToutesLesQualites);

outln(proposition);
```

Sortie :

```
de toutes les qualités, le courage est la plus importante
```

L'ellipse d'un syntagme en entier est plus simple, comme le montre l'exemple 4.2.

Exemple 4.2: Utilisation du trait `ELIDED` pour l'ellipse d'un syntagme.

```
NPPhraseSpec jeanPierre = factory.createNounPhrase("Jean-Pierre");
NPPhraseSpec moi = factory.createNounPhrase("moi");
CoordinatedPhraseElement sujet =
     factory.createCoordinatedPhrase(jeanPierre, moi);

SPhraseSpec texto = factory.createClause(sujet, "partir");
texto.setFeature(Feature.TENSE, Tense.FUTURE);

NPPhraseSpec train = factory.createNounPhrase("le", "train");
train.addPreModifier("premier");
PPPhraseSpec moyenDeTransport =
     factory.createPrepositionPhrase("par", train);
texto.addComplement(moyenDeTransport);

sujet.setFeature(Feature.ELIDED, true);

outln(texto);
```

Sortie :

```
partirons par le premier train
```

5 Agrégation de propositions

SimpleNLG comprend un module servant à faire l'agrégation de propositions. (*package* `simplenlg.aggregation`) Il est possible de faire l'agrégation de propositions partageant le même sujet ou le même syntagme verbal. L'exemple 5.1 montre l'utilisation d'une des règles d'agrégation. Celles-ci prennent en entrée une liste de NLGElement et en retournent une autre. Si une agrégation est possible, elle est effectuée et la liste retournée est plus courte. Dans l'exemple ci-dessous, on a trois propositions qui agrégées en une seule. Il faut récupérer le premier élément de la liste (qui n'en contient qu'un) avant de le réaliser de la manière habituelle.

Exemple 5.1: Agrégation de propositions ayant le même syntagme verbal.

```
WordElement Marie = lexicon.getWord("Marie", LexicalCategory.NOUN);
Marie.setFeature(LexicalFeature.GENDER, Gender.FEMININE);
Marie.setFeature(LexicalFeature.PROPER, true);
WordElement Julie = lexicon.getWord("Julie", LexicalCategory.NOUN);
Julie.setFeature(LexicalFeature.GENDER, Gender.FEMININE);
Julie.setFeature(LexicalFeature.PROPER, true);
WordElement Martin = lexicon.getWord("Martin", LexicalCategory.NOUN);
Martin.setFeature(LexicalFeature.GENDER, Gender.MASCULINE);
Martin.setFeature(LexicalFeature.PROPER, true);
NLGElement proposition1 = factory.createClause(Marie, "devenir",
   "gentil");
NLGElement proposition2 = factory.createClause(Julie, "devenir",
   "gentil");
NLGElement proposition3 = factory.createClause(Martin, "devenir",
   "gentil");
proposition1.setFeature(Feature.TENSE, Tense.PAST);
proposition2.setFeature(Feature.TENSE, Tense.PAST);
proposition3.setFeature(Feature.TENSE, Tense.PAST);

List<NLGElement> elements = Arrays.asList(proposition1,  proposition2,
   proposition3);
ClauseCoordinationRule clauseCoord = new ClauseCoordinationRule();
List<NLGElement> result = clauseCoord.apply(elements);
NLGElement aggregated = result.get(0);
outln(aggregated);
```

Sortie :

```
Marie, Julie et Martin sont devenus gentils
```

6 Formatage du document

Le formatage du document se fait exactement de la même façon en anglais et en français. Les éléments de document sont représentés par des objets de la classe `DocumentElement`. Ils sont de six types : phrase, paragraphe, item de liste, liste, section, document. Ils sont créés avec les méthodes suivantes de la *factory*, dans le même ordre :

- createSentence(...)
- createParagraph(...)
- createListItem(...)
- createList(...)
- createSection(...)
- createDocument(...)

Une proposition (`SPPhraseSpec`) doit être contenue dans une phrase (*sentence*) ou dans une autre proposition pour être correctement formatée. L'exemple 6.1 illustre la réalisation d'une phrase isolée à l'aide de la méthode `realiseSentence(...)`. Cette méthode s'occupe automatiquement d'inclure ce qu'elle reçoit en argument dans un objet de catégorie `DocumentCategory.SENTENCE`. Une majuscule est mise au début de la phrase et un point à la fin.

Exemple 6.1: Réalisation d'une phrase isolée avec majuscule de début de phrase et point final.

```
NPPhraseSpec snMaison = factory.createNounPhrase("le", "maison");
SPhraseSpec elleVendreMaison = factory.createClause("elle", "vendre",
   snMaison);
String phrase1 = realiser.realiseSentence(elleVendreMaison);
System.out.println(phrase1);
```

Sortie :

```
Elle vend la maison.
```

Lors de la création d'une section ou d'un document, il est possible de spécifier un titre qui sera imprimé sur une ligne juste avant cet élément. On peut aussi utiliser à cette fin la méthode `setTitle`(...) du `DocumentElement`.

Une ligne vide est imprimée après chaque paragraphe. Il faut par contre inclure dans le titre les lignes vides qu'on veut voir apparaître avant et après celui-ci.

L'exemple 6.2 illustre l'utilisation des éléments de document. On peut aussi y voir un exemple de création d'une anaphore pronominale, par la copie d'un syntagme nominal et l'ajout du trait `Feature.PRONOMINAL`.

Exemple 6.2: Utilisation des éléments de document et anaphore pronominale.

```
NPPhraseSpec snLeChien = factory.createNounPhrase("le", "chien");
NPPhraseSpec snUnOs = factory.createNounPhrase("un", "os");
NPPhraseSpec snLeCiel = factory.createNounPhrase("le", "ciel");
NPPhraseSpec snDesNuages = factory.createNounPhrase("un", "nuage");
snDesNuages.setPlural(true);
NPPhraseSpec snMartin = factory.createNounPhrase("Martin");
NPPhraseSpec snMartinPronom = factory.createNounPhrase(snMartin);
snMartinPronom.setFeature(Feature.PRONOMINAL, true);
SPhraseSpec proposition1 = factory.createClause(
   snLeChien, "ronger", snUnOs);
SPhraseSpec proposition2 = factory.createClause(
   snMartin, "appeler", snLeChien);
// La proposition 3 est créée directement sous forme de phrase
// (ci-dessous), pour illustrer cette possibilité.
SPhraseSpec proposition4 = factory.createClause(
   snMartin, "regarder", snLeCiel);
SPhraseSpec proposition5 = factory.createClause(
   snMartinPronom, "voir", snDesNuages);
proposition5.addComplement("y");
CoordinatedPhraseElement coord =
   factory.createCoordinatedPhrase(proposition4, proposition5);

DocumentElement phrase1 = factory.createSentence(proposition1);
DocumentElement phrase3 = factory.createSentence(
   snLeChien, "regarder", snMartinPronom);
DocumentElement phrase4 = factory.createSentence(
   "le chien semble considérer longuement la question");
DocumentElement phrase5 = factory.createSentence(coord);

DocumentElement paragraphe1 = factory.createParagraph(phrase1);
DocumentElement paragraphe2 = factory.createParagraph();
paragraphe2.addComponent(proposition2);
paragraphe2.addComponent(phrase3);
paragraphe2.addComponent(phrase4);
DocumentElement paragraphe3 = factory.createParagraph(phrase5);

List<DocumentElement> listePara = Arrays.asList(paragraphe1,paragraphe2);
DocumentElement section1 = factory
   .createSection("\nTitre de la section 1\n", listePara);
DocumentElement section2 = factory
   .createSection("\nTitre de la section 2\n", paragraphe3);

DocumentElement document = factory.createDocument("\nTitre du document\n",
   section1);
document.addComponent(section2);

String realised = realiser.realise(document).getRealisation();
System.out.print(realised);
```

Sortie (exemple 6.2) :

```
Titre du document


Titre de la section 1

Le chien ronge un os.

Martin appelle le chien. Le chien le regarde. Le chien semble considérer
longuement la question.


Titre de la section 2

Martin regarde le ciel et il y voit des nuages.
```

7 Réalisation bilingue

Avec SimpleNLG-EnFr, on peut non seulement réaliser du texte en anglais ou en français, mais aussi réaliser du texte dans les deux langues dans le même document, ou même dans la même phrase.

Dans l'exemple 7.1, un ensemble de correspondances est établi entre des sens lexicaux, d'une part, et des mots anglais, puis des mots français, d'autre part (p. 55-56). Une même structure syntaxique est ensuite construite (p. 57) en employant l'un, puis l'autre ensemble de correspondances pour y insérer les mots. Finalement, les deux structures sont réalisées dans le même document (ci-dessous), ce qui donne le résultat présenté à la page 58.

Exemple 7.1: Réalisation bilingue d'une même structure syntaxique.

```
public static void main(String[] args) {
   // Chargement du lexique et de la factory anglais.
   Lexicon englishLexicon = new simplenlg.lexicon.english.XMLLexicon();
   NLGFactory englishFactory = new NLGFactory(englishLexicon);

   // Chargement du lexique et de la factory français.
   Lexicon frenchLexicon = new simplenlg.lexicon.french.XMLLexicon();
   NLGFactory frenchFactory = new NLGFactory(frenchLexicon);

   Realiser realiser = new Realiser();

   // Création des correspondances sens-mots en anglais et en français.
   Map<DictioEntry,WordElement> englishSensesMapping =
        createEnglishSensesMapping(englishLexicon);
   Map<DictioEntry,WordElement> frenchSensesMapping =
        createFrenchSensesMapping(frenchLexicon);

   // Construction de la structure d'une phrase en anglais et en français.
   SPhraseSpec englishGreeting =
        buildGreetCrowd(englishFactory, englishSensesMapping);
   SPhraseSpec frenchGreeting =
        buildGreetCrowd(frenchFactory, frenchSensesMapping);

   // Réalisation des deux structures dans le même document.
   DocumentElement paragraph = englishFactory.createParagraph();
   paragraph.addComponent(englishGreeting);
   paragraph.addComponent(frenchGreeting);
   DocumentElement document =
        englishFactory.createDocument("Bilingual greeting\n");
   document.addComponent(paragraph);

   String outString = realiser.realise(document).getRealisation();
   System.out.print(outString);
}
```

Exemple 7.1 (suite)

```
public static Map<DictioEntry, WordElement> createEnglishSensesMapping(
          Lexicon englishLexicon) {
     Map<DictioEntry,WordElement> englishSensesMapping =
          new EnumMap<DictioEntry,WordElement>(DictioEntry.class);

     englishSensesMapping.put(DictioEntry.ADULT_MALE,
                    englishLexicon.getWord("man", LexicalCategory.NOUN));
     englishSensesMapping.put(DictioEntry.BILINGUAL,
          englishLexicon.getWord("bilingual", LexicalCategory.ADJECTIVE));
     englishSensesMapping.put(DictioEntry.CROWD,
          englishLexicon.getWord("crowd", LexicalCategory.NOUN));
     englishSensesMapping.put(DictioEntry.DEFINITE_DETERMINER,
          englishLexicon.getWord("the", LexicalCategory.DETERMINER));
     englishSensesMapping.put(DictioEntry.GREETING_VERB,
          englishLexicon.getWord("greet", LexicalCategory.VERB));
     englishSensesMapping.put(DictioEntry.LANGUAGE_PREPOSITION,
          englishLexicon.getWord("in", LexicalCategory.PREPOSITION));
     englishSensesMapping.put(DictioEntry.OLD,
          englishLexicon.getWord("old", LexicalCategory.ADJECTIVE));
     englishSensesMapping.put(DictioEntry.SPANISH_LANGUAGE,
          englishLexicon.getWord("Spanish", LexicalCategory.NOUN));
     englishSensesMapping.put(DictioEntry.YET,
          englishLexicon.getWord("yet", LexicalCategory.ADVERB));

     return englishSensesMapping;
}
```

Exemple 7.1 (suite)

```
private static Map<DictioEntry, WordElement> createFrenchSensesMapping(
          Lexicon frenchLexicon) {
     Map<DictioEntry,WordElement> frenchSensesMapping =
          new EnumMap<DictioEntry,WordElement>(DictioEntry.class);

     frenchSensesMapping.put(DictioEntry.ADULT_MALE,
          frenchLexicon.getWord("homme", LexicalCategory.NOUN));
     frenchSensesMapping.put(DictioEntry.BILINGUAL,
          frenchLexicon.getWord("bilingue", LexicalCategory.ADJECTIVE));
     frenchSensesMapping.put(DictioEntry.CROWD,
          frenchLexicon.getWord("foule", LexicalCategory.NOUN));
     frenchSensesMapping.put(DictioEntry.DEFINITE_DETERMINER,
          frenchLexicon.getWord("le", LexicalCategory.DETERMINER));
     frenchSensesMapping.put(DictioEntry.GREETING_VERB,
          frenchLexicon.getWord("saluer", LexicalCategory.VERB));
     frenchSensesMapping.put(DictioEntry.LANGUAGE_PREPOSITION,
          frenchLexicon.getWord("en", LexicalCategory.PREPOSITION));
     frenchSensesMapping.put(DictioEntry.OLD,
          frenchLexicon.getWord("vieux", LexicalCategory.ADJECTIVE));
     frenchSensesMapping.put(DictioEntry.SPANISH_LANGUAGE,
          frenchLexicon.getWord("espagnol", LexicalCategory.NOUN));
     frenchSensesMapping.put(DictioEntry.YET,
          frenchLexicon.getWord("encore", LexicalCategory.ADVERB));

     return frenchSensesMapping;
}
```

Exemple 7.1 (suite)

```
public static SPhraseSpec buildGreetCrowd(NLGFactory factory,
          Map<DictioEntry, WordElement> wordSensesMapping) {

     WordElement definiteDeterminer =
          wordSensesMapping.get(DictioEntry.DEFINITE_DETERMINER);
     WordElement adultMale =
          wordSensesMapping.get(DictioEntry.ADULT_MALE);
     WordElement old = wordSensesMapping.get(DictioEntry.OLD);
     WordElement bilingual = wordSensesMapping.get(DictioEntry.BILINGUAL);
     WordElement greetingVerb =
          wordSensesMapping.get(DictioEntry.GREETING_VERB);
     WordElement crowd = wordSensesMapping.get(DictioEntry.CROWD);
     WordElement yet = wordSensesMapping.get(DictioEntry.YET);
     WordElement in =
          wordSensesMapping.get(DictioEntry.LANGUAGE_PREPOSITION);
     WordElement spanish =
          wordSensesMapping.get(DictioEntry.SPANISH_LANGUAGE);

     NPPhraseSpec theMan = factory.createNounPhrase(definiteDeterminer,
          adultMale);
     theMan.addModifier(old);
     theMan.addModifier(bilingual);

     NPPhraseSpec theCrowd =
          factory.createNounPhrase(definiteDeterminer, crowd);
     theCrowd.setFeature(Feature.PRONOMINAL, true);

     SPhraseSpec greeting =
          factory.createClause(theMan, greetingVerb, theCrowd);

     greeting.setFeature(Feature.NEGATED, true);
     greeting.setFeature(Feature.TENSE, Tense.PAST);

     greeting.addPreModifier(yet);

     PPPhraseSpec inSpanish =
          factory.createPrepositionPhrase(in, spanish);
     greeting.addModifier(inSpanish);

     return greeting;
}
```

Sortie (exemple 7.1) :

```
Bilingual greeting

The old, bilingual man did not yet greet it in Spanish. Le vieil homme
bilingue ne l'a pas encore saluée en espagnol.
```

On remarque dans la sortie de l'exemple 7.1 que les adjectifs épithètes ont été correctement placés par défaut à gauche ou à droite du nom auquel ils se rapportent. Également, les auxiliaires et les adverbes de négation sont placés correctement et la flexion et les règles morphophonologiques sont bien appliquées. En particulier, le participe passé avec *avoir* s'accorde avec son objet, *l'* (remplaçant *la foule*), car ce dernier est placé avant lui. L'adverbe a été placé comme demandé comme prémodificateur. Par défaut, *yet* aurait été placé en tête de phrase et *encore* aurait été placé devant *en espagnol*.

Dans l'exemple 7.2, ci-dessous, une phrase est construite en mêlant mots et structures syntaxiques des deux langues. (La phrase produite serait un mélange de *Les hommes conduisent une auto.* et *The men drive a car.* On a ici évité de répéter la définition des lexiques, des *factories* et du réalisateur, qui sont les mêmes que dans l'exemple 7.1.) La sortie de cet exemple est bien sûr plutôt invraisemblable, mais a la vertu d'illustrer le fonctionnement de SimpleNLG-EnFr. On voit que l'accord est fait entre le sujet et le verbe et entre le déterminant et le nom, que la flexion s'effectue correctement et qu'une règle morphophonologique est appliquée (*a* → *an* devant voyelle). Le tout en se référant à la langue du syntagme ou du mot concerné.

Exemple 7.2: Phrase mêlant des mots français et anglais.

```
WordElement detLe =
   frenchLexicon.lookupWord("le", LexicalCategory.DETERMINER);
WordElement detA =
   englishLexicon.lookupWord("a", LexicalCategory.DETERMINER);

NPPhraseSpec npLeDog = englishFactory.createNounPhrase(detLe, "man");
npLeDog.setPlural(true);
NPPhraseSpec snAOs = frenchFactory.createNounPhrase(detA, "auto");
SPhraseSpec proposition =
   frenchFactory.createClause(npLeDog, "conduire", snAOs);

String realised = realiser.realiseSentence(proposition);
System.out.println(realised);
```

Sortie :

```
Les men conduisent an auto.
```

8 Conclusion

8.1 Bilan

L'objectif d'adapter SimpleNLG pour qu'il puisse faire la réalisation de textes en français comme en anglais a été atteint. SimpleNLG-EnFr peut réaliser du texte mêlant les deux langues dans le même document. La réalisation en anglais s'effectue exactement comme dans SimpleNLG v4.2. Pour la réalisation en français, SimpleNLG-EnFr 1.1 satisfait à presque toutes les recommandations du *Français fondamental (1er Degré)* pour ce qui est de la couverture grammaticale. Un lexique français de 3871 entrées permet de couvrir les mots les plus courants.

Pour ce qui est de l'utilisation de la bibliothèque, l'interface programmeur (API) a pu être conservée pratiquement intacte. De plus, une documentation française détaillée a été rédigée.

Toutefois, une partie du fonctionnement interne de SimpleNLG a dû être modifié. En effet, il a fallu réorganiser certaines parties du code source original pour pouvoir isoler ce qui était spécifique à l'anglais de ce qui était plus générique. Ce n'est qu'après que la grammaire française a pu être ajoutée. À ce jour, environ un tiers des classes de SimpleNLG v4.2 ont dû être modifiées; 77 méthodes ont ainsi été modifiées et 61 ajoutées aux classes existantes. Une dizaine de *packages* et une trentaine de classes ont été créés, de même que 37 nouveaux traits (*features*).

Les tests JUnit ont été conservés. On s'est assurés que SimpleNLG-EnFr les passe toujours avec succès et on a ajouté une nouvelle batterie de tests pour le français en se basant sur les tests anglais.

8.2 Travaux futurs

Pour terminer la couverture du *Français fondamental (1er Degré)*, il ne restera principalement qu'à ajouter le traitement complet de la représentation des nombres en adjectifs cardinaux et ordinaux. Par contre, l'ajout des interrogatives avec inversion du sujet ne semble pas nécessaire, car les phrases équivalentes avec l'introducteur *est-ce que* sont déjà couvertes. De même, l'ajout de l'indicatif passé simple et passé antérieur et du subjonctif imparfait et plus-que-parfait n'est pas envisagé, leur utilité pratique n'étant pas jugée suffisante.

Pour enrichir le lexique XML français par défaut, on pourrait entre autres y ajouter l'échelon 100 de l'échelle orthographique Dubois Buyse. Il s'agit de 213 mots fréquents qui ont été ajoutés à cette échelle après coup pour la mettre à jour (Simonet), mais qui n'étaient pas inclus dans le fichier fourni par Bacquet sur lequel est basé le lexique actuel. Ils sont toutefois disponibles sur le site web de Deleuze. Des lexiques français spécialisés complémentaires pourraient aussi être mis à disposition sous forme de fichiers XML ou rendus accessibles par une nouvelle classe.

En outre, il faudrait intégrer à SimpleNLG-EnFr, qui est basé sur la version 4.2 de SimpleNLG, les nouveautés de la version 4.3 (ou ultérieure), tel que le module de réalisation XML. Ce module permet d'entrer des structures de SimpleNLG sous forme de fichier XML d'un type défini, plutôt que sous forme d'objets Java.

Maintenant qu'une version bilingue de SimpleNLG a été créée, il devrait être plus facile d'ajouter d'autres langues proches de l'anglais et du français. Ce qui est commun à ces deux langues a en effet été séparé de ce qui leur est spécifique. Il ne resterait donc grosso modo qu'à ajouter ce qui est spécifique à l'éventuelle troisième langue, si elle ressemble suffisamment aux deux premières.

9 Références

BACQUET, Olivier. *Echelle orthographique Dubois Buyse*, lien *format texte délimité*, [http://o.bacquet.free.fr/db2.htm; http://o.bacquet.free.fr/duboisbuyse_txt.zip] (site consulté le 14 juillet 2011).

CNRTL. *CNRTL : Centre National de Ressources Textuelles et Lexicales – Morphalou*, bouton *Télécharger Morphalou 2.0*, [http://www.cnrtl.fr/lexiques/morphalou/] (site consulté le 14 juillet 2011).

DELEUZE, Jean-Marc. *Recherches sur l'Echelle d'acquisition de l'orthographe lexicale*, [http://jeanmarc.deleuze.free.fr/echelleDB/recherche.html] (site consulté le 5 juillet 2012).

GREVISSE, Maurice (1993). *Le bon usage, grammaire française*, 12e édition refondue par André Goosse, 8e tirage, Éditions Duculot, Louvain-la-Neuve, Belgique.

MANSOURI, Mohammed Issaoui (1996). *Le Mansouris, tous les verbes usuels entièrement conjugués et orthographiés*. CAPT, Éditeurs, Montréal, Canada.

Ministère de l'Éducation nationale, Direction de la Coopération avec la Communauté et l'Étranger (France) (1959). *Le français fondamental (1er Degré)*, Publication de l'Institut Pédagogique National, Paris, France.

REITER, Ehud, et al. *A tutorial for SimpleNLG (version 4)*, [http://code.google.com/p/simplenlg/wiki/Tutorial] (site consulté le 27 juillet 2011)

SIMONET, Émile. *L'échelle Dubois Buyse en deux mots*, [http://emile.simonnet.free.fr/dubois/aideDB.htm] (site consulté le 5 juillet 2012).

Index des exemples

Annexe 1 : Exemples du tutoriel de la version originale de SimpleNLG

Le tutoriel de la version originale de SimpleNLG, pouvant être trouvé à l'adresse http://code.google.com/p/simplenlg/wiki/Tutorial, peut servir d'introduction à l'approche générale de SimpleNLG. Il y a peu de changements à apporter pour adapter le code des exemples de ce tutoriel à la version bilingue. Voici les remplacements à effectuer :

```
import simplenlg.realiser.english.*;
Realiser realiser = new Realiser(lexicon);
```

sont remplacés par :

```
import simplenlg.realiser.*;
Realiser realiser = new Realiser();
```

À la section IV, pour charger un lexique XML autre que le lexique XML anglais par défaut, on doit choisir la classe XMLLexicon provenant du package désiré, selon la langue.

```
Lexicon lexicon = new XMLLexicon("my-lexicon.xml");
```

est remplacé par :

```
Lexicon frenchLexicon = new simplenlg.lexicon.english.XMLLexicon("my-english-lexicon.xml");
```

ou

```
Lexicon englishLexicon = new simplenlg.lexicon.french.XMLLexicon("my-french-lexicon.xml");
```

Le lexique XML français par défaut est chargé de cette façon :

```
Lexicon defaultFrenchLexicon = new simplenlg.lexicon.french.XMLLexicon();
```

Annexe 2 : Fonction outln (NLGElement) utilisée dans les exemples

La fonction suivante est utilisée dans beaucoup d'exemples de ce document en guise de raccourci d'écriture. (Elle ne fait pas partie de l'API de SimpleNLG, ni de celle de SimpleNLG-EnFr.) Cette fonction n'est pas utilisée dans les exemples 6.2 et 7.1 pour souligner qu'il n'est pas nécessaire, dans ces deux cas, d'ajouter un changement de ligne final, celui-ci étant déjà prévu dans le formatage du dernier paragraphe du texte réalisé.

Remarquez ci-dessous les types de retour des méthodes de SimpleNLG utilisées pour la réalisation. Également, remarquez que le constructeur du réalisateur de SimpleNLG-EnFr ne prend pas de lexique en argument, contrairement à celui de SimpleNLG.

```
public static final Realiser realiser = new Realiser();

public static void outln(NLGElement outElement) {
     NLGElement realisedElement = realiser.realise(outElement);
     String realisation = realisedElement.getRealisation();
     System.out.println(realisation);
}
```

Ou sous forme abrégée :

```
public static final Realiser realiser = new Realiser();

public static void outln(NLGElement outElement) {
     System.out.println(realiser.realise(outElement).getRealisation());
}
```

Annexe 3 : Licence de SimpleNLG-EnFr

SimpleNLG-EnFr est distribué sous la même licence que SimpleNLG v4.2, conformément aux exigences de cette licence. Il s'agit de la Mozilla Public licence (MPL) version 1.1. Voici ci-dessous l'entête placée au début de chaque fichier de code source de SimpleNLG-EnFr.

En résumé, vous pouvez redistribuer SimpleNLG-EnFr en l'ayant modifié ou non, à condition de le faire sous la même licence. Si vous faites des modifications, elles doivent être répertoriées et le code source doit être disponible. Veuillez vous référer à http://www.mozilla.org/MPL/ pour le texte exact de la licence.

```
The contents of this file are subject to the Mozilla Public License
Version 1.1 (the "License"); you may not use this file except in
compliance with the License. You may obtain a copy of the License at
http://www.mozilla.org/MPL/

Software distributed under the License is distributed on an "AS IS"
basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the
License for the specific language governing rights and limitations
under the License.

The Original Code is "Simplenlg".

The Initial Developer of the Original Code is Ehud Reiter, Albert Gatt and
Dave Westwater.
Portions created by Ehud Reiter, Albert Gatt and Dave Westwater are
Copyright (C) 2010-11 The University of Aberdeen. All Rights Reserved.

Contributor(s): Ehud Reiter, Albert Gatt, Dave Wewstwater, Roman Kutlak,
Margaret Mitchell, Pierre-Luc Vaudry.
```