JSrealB : Approche systématique pour la réalisation multilingue de textes

Paul Molins

Département informatique
INSA de Lyon
2014/2015

Sous la responsabilité de :
Guy Lapalme : Professeur au Département d'informatique et de recherche opérationnelle, Université de Montréal
Sylvie Calabretto : Enseignante au Département Informatique, INSA de Lyon

Résumé. Lorsqu'un service est destiné à des utilisateurs non experts, l'affichage de données brutes ne suffit pas. C'est le cas pour les données météorologiques qui doivent être triées et expliquées. Dans ce rapport, nous présentons JSrealB, un réalisateur de textes en anglais et français qui sera intégré dans l'interface de présentation de données météorologiques MeteoVis. Ce réalisateur fonctionne par l'application de deux types de règles. Il y a les règles systématiques exprimées sous forme de tables notamment pour les flexions et les règles algorithmiques de propagation pour les accords. Cette approche systématique permet de réaliser des textes dans les deux langues et d'en supporter d'autres avec un minimum de développements spécifiques.

Mots-clés : Réalisateur de textes bilingue, Réalisateur de textes web, Génération de prévisions météorologiques

Abstract. When a service is targeted towards non-expert users, displaying raw data is not enough. This is the case for meteorological data which must be sorted and explained. In this report, we present JSrealB, an English and French text realizer which will be integrated into MeteoVis, a meteorological data display interface. The realisation engine applies two types of rules. There are systematic rules expressed as tables for inflection and algorithmic propagation rules for agreements. This systematic approach is compatible with texts in both languages and it is possible to support other with few specific developments.

Keywords: Bilingual text realizer, Web text realizer, Weather forecast text generation

1 Introduction

La génération automatique de textes est généralement utilisée pour rendre lisibles des données brutes par un humain. Ces techniques sont aujourd'hui exploitées par l'industrie et des sociétés s'en sont fait une spécialité. Les applications sont très variées, récemment, le journal « Le Monde » a expérimenté les services de Syllabs[1] lors des élections départementales françaises de 2015, l'utilisation de ce logiciel a permis la génération automatique de courts textes de présentation des résultats des élections de chaque canton, plus attrayants que des tableaux de données, et comme le souligne cet édito du journal « Le Monde » (Bronner, 2015), plus efficace pour le référencement des articles par les moteurs de recherche. La génération automatique de textes est bel et bien un enjeu actuel.

Ces outils sont également très utilisés dans le domaine de la météorologie, pour générer des prévisions météorologiques. Scribe 3.0, créé en 1997 (Verret, et al., 1997), et déployé en 2004 sur les services météorologiques d'Environnement Canada[2] pour calculer les données prévisionnelles (température, vent, couverture nuageuse, etc.), il produit également des prévisions textuelles en français et en anglais. C'est également l'objectif de ce projet, intégrer des prévisions météorologiques textuelles dans MeteoVis (Mouine and Lapalme, 2014). MeteoVis est une interface de présentation de données météorologiques, dans laquelle M. Mouine illustre des concepts de présentation développés lors de sa thèse en 2014. L'interface propose un aperçu des conditions courantes dans la ville canadienne choisie, les prévisions à 7 jours sous forme de graphique et des prévisions textuelles générées automatiquement à partir des données brutes et mettant en avant les événements importants de la journée. MeteoVis est un projet gouvernemental canadien, or au Canada, l'anglais et le français sont les langues officielles, les prévisions doivent donc être émises dans les deux langues.

La génération automatique de textes est une discipline encore réservée à des sociétés spécialisées étant donné que les outils sont complexes à installer et utiliser. La génération automatique de textes consiste à réaliser une représentation formelle du contenu à partir de données brutes, c'est-à-dire, un texte syntaxiquement et sémantiquement correct (Reiter and Dale, 2000). Ce processus se décompose en plusieurs étapes, de la détermination du contenu à la réalisation de textes qui est la dernière étape. Elle vise à créer le texte en respectant les règles de syntaxe, morphologie et d'orthographe, à partir d'une représentation syntaxique.

JSreal (Daoust and Lapalme, 2014) est un réalisateur de textes pour la programmation web, écrit en JavaScript, qui propose une nouvelle approche permettant de simplifier l'utilisation de ce type d'outils. JSreal peut, en principe, être utilisé sans connaissances informatiques, la représentation syntaxique reposant sur les notations classiques des grammaires en constituants. En effet, un nom est identifié par l'abréviation `N` pour l'anglais `Noun`, un déterminant par `D`. De même pour les syntagmes[3], `NP` signifie `Noun Phrase` (soit le syntagme nominal), `VP` pour le syntagme verbal et `S` est utilisé pour unifier les syntagmes sous forme de phrase. La Figure 1 montre un exemple de représentation syntaxique pour JSrealB avec la mise au pluriel du syntagme nominal par l'utilisation de `n("p")` où `n` désigne le nombre grammatical et la valeur `p` pour le pluriel et `t` permet d'identifier le temps qui est le passé avec `ps`.

```
S(
   NP( D("a"), N("woman") ).n("p"),
   VP("eat").t("ps")
)
```

The women ate.

Figure 1 Spécification d'un texte avec JSrealB et la réalisation correspondante

Pour le projet MeteoVis, JSreal semble être la solution idéale pour générer les bulletins météorologiques. Une solution simple à installer et facile à utiliser. Néanmoins, JSreal est un outil développé pour réaliser des textes en français uniquement. Les règles de cette langue étant codifiées directement dans l'algorithme, une réingénierie pour porter le programme vers le bilinguisme serait très

[1] Syllabs est une société française qui développe notamment des solutions comme Data2Content qui génère du texte à partir de données brutes. [http://www.syllabs.com]

[2] Environnement Canada est le Ministère canadien de l'Environnement.

[3] Une phrase peut être segmentée en syntagmes qui forment une unité syntaxique composée d'éléments terminaux.

coûteuse, puisque la réutilisation de composants est très limitée. De même, installer une solution comme SimpleNLG (Gatt and Reiter, 2009) ou la version bilingue français/anglais SimpleNLG-EnFr (Vaudry and Lapalme, 2013), écrite en Java, nécessiterait une infrastructure informatique, le développement de services, une installation plus longue et complexe pour générer de courts textes. La Figure 2 présente la même réalisation que la Figure 1, mais avec SimpleNLG-EnFr.

```
NPPhraseSpec subject = nlgFactory.createNounPhrase("the", "woman");
subject.setPlural(true);
SPhraseSpec sentence = nlgFactory.createClause(subject, "eat");
sentence.setFeature(Feature.TENSE, Tense.PAST);
realiser.realiseSentence(sentence);
```

The women ate.

Figure 2 Réalisation du texte équivalent à celui de la Figure 1 avec SimpleNLG

Nous avons opté pour le développement d'une solution multilingue, pour le web, écrite en JavaScript, et reprenant la représentation syntaxique définie par JSreal. Ce nouvel outil a été baptisé JSrealB. De par son mode de fonctionnement, il est conçu pour réaliser des textes dans d'autres langues flexionnelles (espagnol, italien, roumain, etc.) quoique dans ce projet nous nous limitons à l'anglais et au français.

Ce projet de fin d'études a été réalisé seul sous la direction de G. Lapalme. Les contributions peuvent être fondées sur des travaux existants comme nous le préciserons par la suite.

2 Présentation de MeteoVis

MeteoVis a été conçu comme un prototype destiné à appliquer des concepts de visualisation avancés pour la présentation d'informations météorologiques. Le projet se différencie par une approche graphique, des icônes et un graphique généré à l'aide de la technologie SVG, ainsi qu'une personnalisation de l'affichage réalisé à partir des préférences des utilisateurs similaires, par clustering en prenant en compte son emplacement, sa langue, l'heure de connexion, et la saison de l'année (Mouine and Lapalme, 2012). La proposition de présentation de données météorologiques ayant été convaincante, il a été décidé de poursuivre ce projet pour Environnement Canada. Ce dernier fournit les données météorologiques courantes des villes du Canada ainsi que les prévisions sur 7 jours. Ces données sont actualisées en permanence par les météorologues et Scribe, l'outil de génération des prévisions, à l'aide du réseau de stations de mesures environnementales du pays.

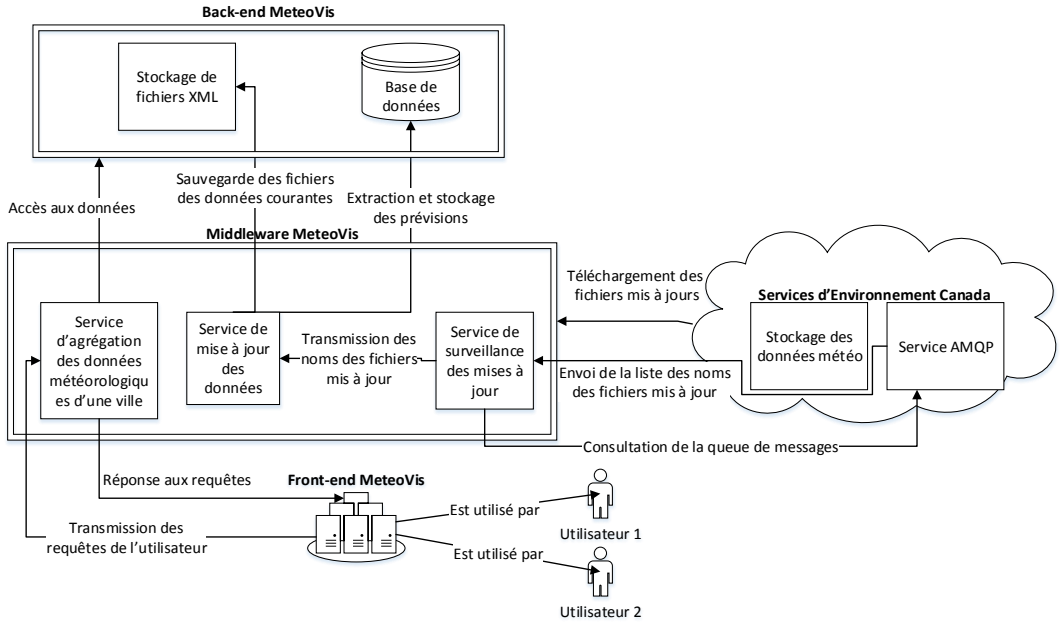

Figure 3 Architecture à 3 couches de MeteoVis après réingénierie

Les deux premiers mois du PFE ont consisté à refondre complètement l'architecture pour transformer le prototype en une application de production pouvant supporter un fort trafic. L'architecture à trois couches est composée d'un serveur front-end pour la présentation des données, d'un middleware pour la gestion des services externes et les différentes opérations sur les données pour extraire les prévisions les plus récentes, et une partie back-end pour le stockage des données [Figure 3].

L'interface utilisateur [Figure 4] présente les données météorologiques courantes (température, pression, visibilité, etc.) de la ville sélectionnée, ainsi que les prévisions pour les 7 prochains jours sous la forme d'un graphique et de prévisions textuelles. Cette interface s'adapte à l'emplacement de l'utilisateur (choix de la ville et gestion des six fuseaux horaires du Canada), à sa langue (français/anglais) et à ses préférences (p. ex. unité de mesure de la température).

La couche applicative utilise le protocole AMQP pour surveiller les mises à jour des fichiers XML fournissant les données de chaque ville canadienne, sur les serveurs d'Environnement Canada. Un message est ajouté à la queue sur le serveur AMQP d'Environnement Canada à chaque fois qu'un fichier est créé ou modifié (Environnement Canada, 2015). Ainsi, il n'est pas nécessaire de consulter l'ensemble des fichiers (plusieurs milliers de nouveaux fichiers chaque jour) pour déterminer ceux qui ont été mis à jour. Une fois les noms des fichiers mis à jour récupérés, le service transmet l'information pour que les données puissent être téléchargées, nettoyées et persistées grâce à la couche d'accès aux données. La couche applicative permet également d'agréger les données à l'aide de la couche d'accès aux données pour répondre aux requêtes des utilisateurs en calculant les prévisions demandées. Ces opérations consistent à trier les mises à jour reçues qui sont partielles et pour une période limitée (p. ex. mise à jour de la vitesse du vent sur un secteur limité entre 10h et 15h, mais également entre 22 et 23h). S'il existe deux valeurs différentes pour une même mesure à la même heure, seule la donnée mise à jour le plus récemment est transmise à la couche de présentation.

La couche d'accès aux données permet de stocker les données que ce soit sous forme d'un fichier XML ou dans une base de données MySQL. Ces données sont stockées pendant toute la durée de leur validité puis supprimées à leur expiration. Ainsi, les services d'accès aux données restent performants, car la quantité de données ne varie quasiment pas.

Grâce à cette architecture, nous sommes capables de présenter aux utilisateurs des prévisions sur 7 jours pour toutes les villes du Canada sans avoir à accéder aux serveurs d'Environnement Canada à chaque consultation de l'interface MeteoVis. Le prototype original consultait systématiquement (à chaque requête d'un utilisateur) les serveurs d'Environnement Canada pour récupérer des informations à jour. Ce système couplé à plusieurs optimisations de l'interface a permis de ramener le temps de chargement des pages à 1,5 seconde en moyenne, alors que le prototype ne répondait qu'en 5 à 30 secondes selon la quantité de données à mettre à jour.

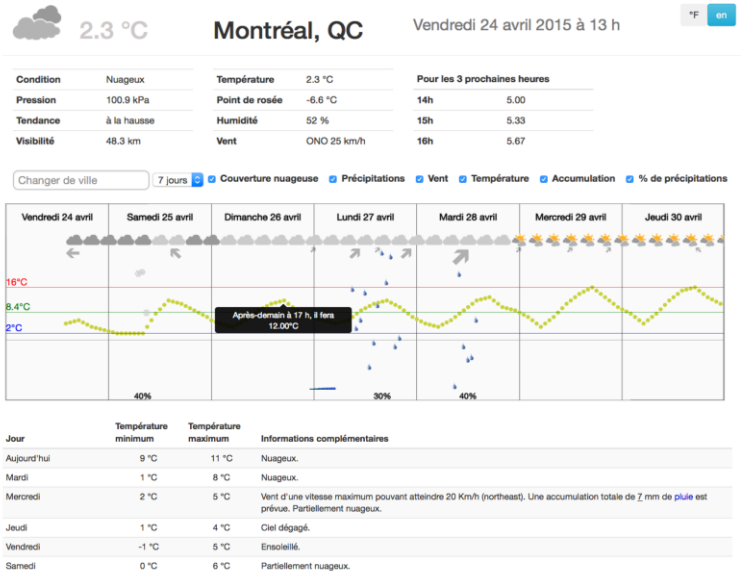

Figure 4 Interface de la nouvelle version de MeteoVis

La dernière étape du projet MeteoVis est la génération automatique de prévisions météorologiques textuelles. Le prototype proposait une solution à base de patterns, c'est-à-dire, de textes à trous, dans les deux langues. Cette solution n'est pas satisfaisante étant donné qu'elle ne prend pas en compte le contexte et qu'elle mélangeait la logique avec l'affichage. De plus, seules les prévisions textuelles journalières étaient disponibles. La version de production de MeteoVis proposera des prévisions sous le graphique de la Figure 4 et sur le graphique sous forme de bulles d'aide au passage de la souris sur les éléments comme l'icône de direction du vent pour indiquer sa vitesse et la direction de manière textuelle (p. ex. *Demain de 8h à 16h, prévoir des vents Sud-Ouest pouvant atteindre 30km/h*). Ces textes d'aide à l'utilisateur seront réalisés à la volée (au passage de la souris). Ainsi, la réalisation s'adapte au contexte (date et valeur de la mesure) avec éventuellement une personnalisation de l'information en fonction des actions de l'utilisateur. Tous ces textes sont réalisés en anglais ou français, selon la langue choisie par l'utilisateur.

3 Présentation de JSrealB

La génération automatique de textes consiste à convertir une représentation machine en un texte en langue naturelle (p. ex. en français) (Reiter and Dale, 2000). Un tel système intervient dans des situations où l'information existe sous forme de données non textuelles. Il requiert tout d'abord une étape de planification : détermination du contenu, structuration (ordre des informations) et choix du lexique. Ensuite vient la réalisation du texte qui consiste à mettre en forme le texte en suivant les règles de syntaxe, morphologie et orthographe. La réalisation de la syntaxe utilise les règles de grammaire pour déterminer la fonction des mots et leur ordre (p. ex. en anglais, l'adjectif précède le nom qu'il qualifie). La réalisation de la morphologie est l'accord des mots selon leur fonction, que ce soit la déclinaison d'un nom, ou la conjugaison du verbe (p. ex. la conjugaison du verbe selon le sujet). La dernière phase est la réalisation orthographique qui va formater la phrase en ajoutant la ponctuation, les majuscules, les contractions et l'élision de certains mots, si nécessaire.

Notre objectif est de proposer un réalisateur capable de réaliser des textes en anglais et français en limitant au maximum les développements spécifiques à chaque langue. Nous avons favorisé la systématisation de l'application de règles en espérant qu'il soit possible de supporter de nouvelles langues à un coût limité.

3.1 Problématique du bilinguisme

Le français et l'anglais sont des langues dont les structures sont semblables. Les deux langues utilisent le même alphabet, ce sont des langues flexionnelles, elles partagent un système de conjugaison similaire (auxiliaire, participe, voix active/passive, temps passé/présent/futur) et la même syntaxe grammaticale Sujet-Verbe-Objet (Shoebottom, 1996). Outre le vocabulaire, des différences de structure existent, notons que l'adjectif ne s'accorde pas en anglais contrairement au français, qu'il se place toujours avant le nom qu'il qualifie en anglais alors qu'en français il peut être avant ou après selon le sens ou l'adjectif (p. ex. *un grand homme* pour un homme remarquable et *un homme grand* pour évoquer sa taille). En français les noms s'accordent en genre et en nombre (p. ex. *un ami* pour le masculin singulier, *des amies* pour le féminin pluriel), en anglais, seulement en nombre (p. ex. *friend* devient *friends* au pluriel).

Ces différences sont prises en compte à plusieurs niveaux. Tout d'abord, les différences syntaxiques et les accords, sont traités au niveau des syntagmes par des algorithmes, ces règles sont issues du Grevisse, *Le bon usage, grammaire française* (Grevisse, 1980).

Ensuite, JSrealB s'appuie sur un lexique spécifique à chaque langue contenant le vocabulaire couvrant l'échelle Dubois-Buyse d'orthographe usuelle française (Ters, et al., 1964) et l'équivalent en anglais. Le lexique peut être complété par l'utilisateur, notamment pour le vocabulaire spécifique à un domaine. Ces lexiques ont été élaborés à partir de ceux de SimpleNLG-EnFr (Vaudry and Lapalme, 2013) et d'un dictionnaire morphologique conçu au RALI, à l'Université de Montréal. Dans les lexiques, les caractéristiques de chaque mot sont renseignées (p. ex. catégorie grammaticale[4], genre grammatical, etc.).

Les caractéristiques contenues dans le lexique sont utilisées pour fléchir correctement les éléments terminaux. En effet, elles permettent de déterminer la bonne terminaison dans les tables de flexion. Ces

[4] La nature d'un mot est nommée *catégorie grammaticale* par Lucien Tesnière.

règles de flexions sont spécifiques à chaque langue, issues d'ouvrages comme le Bescherelle[5] pour la conjugaison (Bescherelle, 2012) et la grammaire (Delaunay and Laurent, 2013). Ces règles sont décrites sous la forme de tables de conjugaisons ou déclinaisons et injectées, avec le lexique, dans le programme.

3.2 Architecture globale

L'architecture de JSreal, développée par N. Daoust, est extrêmement liée à la langue de réalisation, le français. En effet, chaque règle est définie à l'aide de conditions (p. ex. pour la conjugaison, une condition pour chaque temps et une autre pour chaque type de verbe), sans utilisation de patron de conception ou une démarche de programmation par intention, les terminaisons régulières sont également écrites dans le programme, les conjugaisons irrégulières comme celle du verbe *être* sont données dans le lexique, de même pour les déclinaisons. Cette organisation qui mélange la logique métier et le code rend le programme complexe, très peu évolutif, les parties du programme sont très dépendantes les unes des autres et le réalisateur est lié à la langue de réalisation.

Ces différents points ont ne permettent pas à JSreal de passer au bilinguisme par réingénierie, il a fallu repenser entièrement l'architecture et surtout le mécanisme d'application des règles. De là a émergé l'idée d'une codification des règles de transformation des éléments terminaux sous forme de tables. Ce dernier est standardisé ainsi il fonctionne pour toutes les langues flexionnelles. Ensuite, il a une nécessité de rendre un tel programme modulable pour rendre possible l'évolution vers de nouvelles langues. Tous ces modules sont cachés derrière une façade en partie spécifique à chaque langue.

Un utilisateur va utiliser une façade qui permet de définir la représentation syntaxique, source de la réalisation de textes. Comme nous le verrons en 3.3, cette façade traite l'analyse de la syntaxe et de la propagation des caractéristiques entre les syntagmes et à l'intérieur de ceux-ci. Par un système d'héritage, il est possible d'utiliser le même scénario de propagation pour certains syntagmes pour toutes les langues et de définir un scénario particulier à chaque langue pour d'autres syntagmes. De même, pour l'identification des fonctions grammaticales, voire la gestion de l'ordre des mots.

Cette façade va manipuler le noyau de JSrealB qui est décomposé en modules pour offrir une meilleure évolutivité. Étant donné que les opérations de base (conjugaison, déclinaison, ponctuation, etc.) sont communes aux langues à flexion, nous pouvons définir des modules qui seront utilisés par toutes les langues avec des paramètres en entrée qui différeront (notamment le lexique et les tables de règles) [Figure 5].

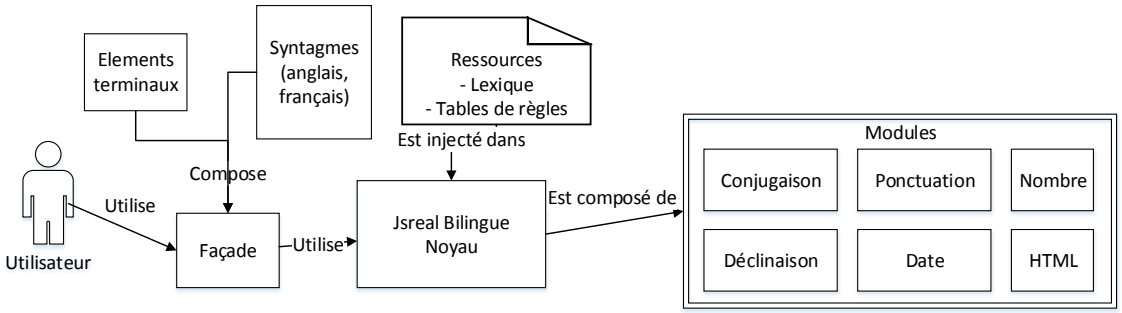

Figure 5 Architecture de JSrealB

3.3 Propagation des caractéristiques

La réalisation d'un texte s'appuie sur la représentation syntaxique hiérarchique sous forme d'arbre [Figure 6]. Cette représentation consiste à décomposer la phrase en syntagmes et en éléments terminaux (les feuilles). Cette identification et les relations entre ces différentes unités lexicales sont le support de la propagation des caractéristiques pour les accords.

La première règle à respecter lors de la réalisation est le respect de la syntaxe, les catégories grammaticales des mots étant précisées dans la représentation syntaxique servant de base à la réalisation,

[5] Collection de livres de référence en grammaire française

il n'est pas nécessaire de les déterminer. Il en va de même pour l'ordre des mots (p. ex. la place de l'adjectif par rapport au nom), si ce n'est quelques exceptions comme le syntagme coordonné pour lequel la place du coordonnant doit être déterminée. Surtout, il faut déterminer la fonction grammaticale de chaque élément de la phrase (p. ex. dans la phrase *Jeanne est à la plage* le sujet est *Jeanne*, le verbe *est* et *à la plage* est un complément circonstanciel de lieu). Les fonctions grammaticales entre le français et l'anglais sont sensiblement similaires. Une fois identifiées, il sera possible d'accorder les éléments de la phrase, c'est-à-dire, de propager les bonnes caractéristiques aux bons éléments en suivant les règles de la langue. Ces règles peuvent être identiques entre le français et l'anglais comme l'accord du verbe avec le sujet ou différentes, l'adjectif s'accorde avec le nom qu'il qualifie en français, de même les articles s'accordent, pas en anglais (p. ex. *une belle fleur* en français, *a beautiful flower* en anglais).

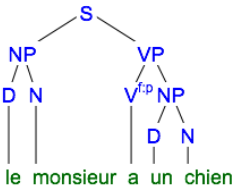

Figure 6 Arbre syntaxique avec JSreal

Le moteur d'application de règles permet d'accorder correctement un élément terminal selon des caractéristiques, encore faut-il les déterminer. Certaines sont données par l'utilisateur comme le temps pour la conjugaison d'un verbe d'autres doivent être transmises d'un syntagme à l'autre comme le syntagme verbal qui doit récupérer, en français et en anglais, la personne à partir du sujet qui peut être un nom ou un pronom (p. ex. conjuguons le verbe « manger » au futur : `S(NP(D(« le »), N(« garçon »)), VP(V(« manger »)).t(« f »))` donnera « Le garçon mangera. »). Ces caractéristiques sont parfois récupérées dans le lexique comme le type du pronom ou encore le genre d'un nom [Figure 7]. Ces propagations suivent les règles d'accord de chaque langue, pour le français, nous nous appuyons sur le Grevisse (Grevisse, 1980).

```
"être": {
    "V": { "tab": "v136" },
    "N": { "g": "m", "tab": ["n3"] }
},
```

Figure 7 Extrait du lexique : élément terminal *être*

De même, en français, dans un syntagme nominal, le déterminant doit s'accorder en genre avec le nom de façon transparente pour l'utilisateur. Ainsi, `NP(D(« le »), N(« voiture »))` donnera « la voiture ». La propagation doit donc se faire à l'intérieur des syntagmes, mais aussi entre les syntagmes.

Enfin, par défaut les syntagmes et les éléments terminaux ont les caractéristiques suivantes :

- Le temps par défaut est le présent
- La personne par défaut est la 3ème
- Le nombre grammatical par défaut est le singulier
- Le genre par défaut est le masculin

3.4 Application de règles

La principale différence entre les langues se situe au niveau des règles à appliquer sur les éléments terminaux, la conjugaison est propre à chaque langue, les accords ne s'appliquent pas aux mêmes catégories (p. ex. les adjectifs s'accordent en français, mais pas en anglais). Il est donc nécessaire de définir ces règles en dehors du programme et de les appliquer de manière systématique.

L'idée est d'appliquer des règles sur les éléments terminaux (ce sont les mots, que ce soit un verbe, un nom ou un déterminant) pour les accorder correctement, de la même manière que le Bescherelle pour la conjugaison, où à chaque verbe correspond une table de terminaisons qu'il faut appliquer au verbe, selon le temps, après avoir retiré la terminaison de l'infinitif. Pour le verbe *parler* (et les autres verbes du premier groupe se conjuguant de la même manière comme *gagner*), il faudra retirer la terminaison `er` de l'infinitif et ajouter la terminaison conjuguée selon le temps (avec `p` pour présent, `i` pour imparfait, etc.)

et la personne (les terminaisons sont ordonnées de la première du singulier à la troisième du pluriel) [Figure 8]. Pour le participe présent `pr` ou passé `pp`, il n'y a qu'une terminaison possible. En français, l'impératif présent `ip` ne se conjugue qu'à trois personnes, la deuxième du singulier, la première et deuxième du pluriel, d'où les valeurs `null` pour signifier que la conjugaison n'existe pas.

```
"v36": {
    "ending": "er",
    "t": {
        "p": ["e","es","e","ons","ez","ent"],
        "i": ["ais","ais","ait","ions","iez","aient"],
        "f": ["erai","eras","era","erons","erez","eront"],
        "ps": ["ai","as","a","âmes","âtes","èrent"],
        "c": ["erais","erais","erait","erions","eriez","eraient"],
        "s": ["e","es","e","ions","iez","ent"],
        "si": ["asse","asses","ât","assions","assiez","assent"],
        "ip": [null,"e",null,"ons","ez",null],
        "pr": "ant",
        "pp": "é"
    }
},
```

Figure 8 Table de conjugaison selon le temps pour les verbes de la classe de *parler*

Nous suivons le même principe pour les déclinaisons. Par exemple, pour les noms que nous voulons accorder en genre et en nombre, nous allons appliquer une terminaison en fonction des caractéristiques souhaitées (p. ex. la terminaison du nom « joueur » au féminin pluriel est *euses* et il faudra retirer la terminaison de base qui est *eur*). Pour accorder l'adjectif *enregistreur* (p. ex. *caisse enregistreuse*), il faut utiliser la même table de déclinaison que pour *joueur* [Figure 9].

```
"n55": {
    "ending": "eur",
    "declension": [ { "val": "eur", "g": "m", "n": "s" },
                    { "val": "euse", "g": "f", "n": "s" },
                    { "val": "eurs", "g": "m", "n": "p" },
                    { "val": "euses", "g": "f", "n": "p" } ]
},
```

Figure 9 Table de déclinaison selon le genre et nombre pour des noms se terminant en *eur*

Enfin, il existe des différences de ponctuation entre les langues flexionnelles. Que ce soit les signes (p. ex. le point d'interrogation inversé en espagnol ¿) ou les règles, d'espacement avant et après le signe de ponctuation (p. ex. il y aura un espace avant et après le point d'exclamation en français, mais seulement après en anglais) ou le caractère entourant d'un signe (p. ex. la question est entourée par ¿ et ? en espagnol alors qu'elle se termine par ? en anglais). Les signes sont définis dans le lexique avec les informations nécessaires comme le complémentaire `compl` pour entourer une unité lexicale [Figure 10]. Les règles sont données sous forme de tables avec l'information de l'espacement avant et après, ainsi que la position d'en le cas d'un signe entourant [Figure 11]. L'emplacement des signes de ponctuations varie aussi d'une langue à l'autre (p. ex. en français la virgule apparait généralement devant la conjonction de coordination *mais*), des règles peuvent être définies et appliquées. Néanmoins, JSrealB ne le fait pas, c'est à l'utilisateur d'indiquer la ponctuation à l'exception de la phrase qui se termine, par défaut, par un point.

```
"(": { "Pc": { "compl": ")", "tab": ["pc5"] } },
")": { "Pc": { "compl": "(", "tab": ["pc6"] } },
```

Figure 10 Extrait de la ponctuation du français dans le lexique

```
"pc5": { "b": " ", "a": "", "pos": "l" },
"pc6": { "b": "", "a": " ", "pos": "r" },
```

Figure 11 Deux règles pour la ponctuation du français

3.5 Exemple de réalisation avec JSrealB

La Figure 13 décrit les différentes étapes de la réalisation de la phrase « Ils découvriront une belle surprise à Noël ! », les chiffres indiquent l'ordre d'exécution des opérations. La représentation syntaxique de départ est celle de la Figure 12.

```
S(
  NP( Pro("je").pe(3).n("p") ),
  VP( V("découvrir"), NP( D("un"), AP( A("beau") ), N("surprise"))),
  PP( P("à"), NP( N("Noël"))
)
```

Figure 12 Représentation syntaxique de l'exemple

La phrase `S` transmet les caractéristiques à ses éléments (p. ex. le temps au syntagme verbal) et déclenche la réalisation de chaque syntagme. La phrase transmet également les caractéristiques permettant la réalisation orthographique (p. ex. mettre en majuscule la première lettre du premier mot). Pour chaque syntagme, une réalisation syntaxique est opérée pour notamment déterminer la fonction grammaticale de chaque élément, avant de les réaliser. Le syntagme nominal se réalise en premier, il transmet les caractéristiques du sujet au syntagme verbal 1.1.3. Le déterminant et l'adjectif du complément d'objet *une belle surprise* doivent s'accorder en genre avec le nom. Cette propriété est remontée après la réalisation du nom *surprise* 2.2.1.2, car contenue dans le lexique, et propagée aux autres éléments du syntagme nominal.

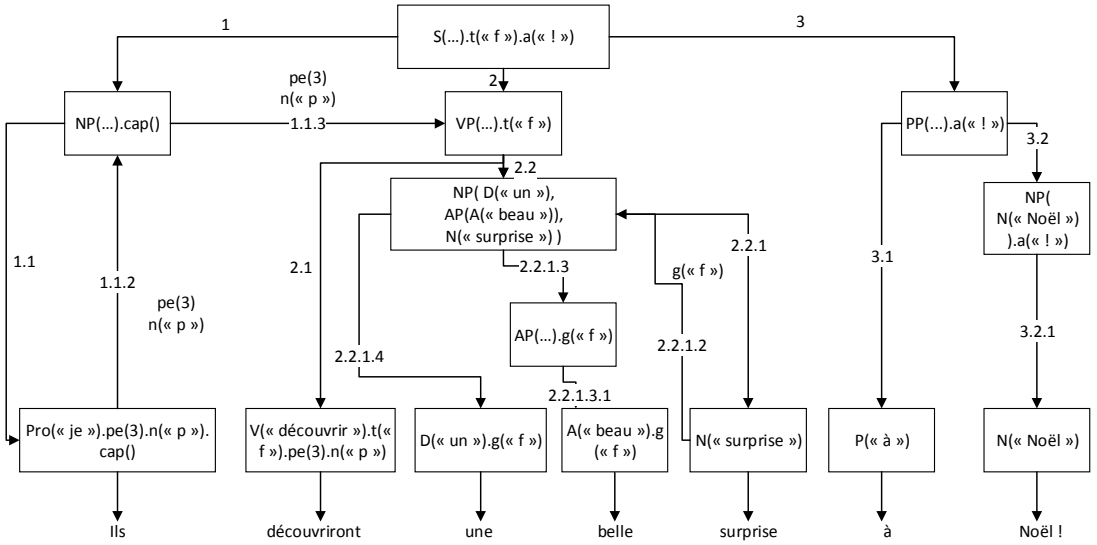

Figure 13 Exemple de réalisation avec JSrealB

3.6 Support de nouvelles langues

L'architecture et le mode de fonctionnement de JSrealB permettent d'envisager le support de nouvelles langues à flexion pour un coût limité. En effet, ce qu'il faut adapter pour supporter une nouvelle langue est d'abord la façade en définissant de nouvelles règles de propagation pour respecter les règles d'accord de la nouvelle langue. Ensuite, il est nécessaire de définir un lexique en respectant le format défini par JSrealB comprenant des mots avec leur catégorie et leurs caractéristiques (genre, nombre, élision, etc.), et la ponctuation. C'est, à n'en pas douter, la partie la plus longue à réaliser. Néanmoins, il est possible de se restreindre au vocabulaire nécessaire pour l'application désirée. Dans notre cas, réaliser des bulletins météorologiques requiert un vocabulaire spécialisé, mais assez limité. Enfin, il faut définir les tables de règles pour les accords et autres transformations. Ces règles sont finalement peu nombreuses et bien connues. La définition de ces tables n'est, en fait, qu'une réécriture de tables du Bescherelle pour la conjugaison en français.

4 Résultats

JSrealB supporte aussi bien l'anglais que le français. Toutes les catégories lexicales sont prises en charge (nom, verbe, pronom, déterminant, adjectif, préposition, conjonction et complément), tous les

syntagmes (nominal, verbal, adjectival, adverbial, prépositionnel, suborné, coordonné), ainsi que la phrase pour unifier une série de syntagmes.

Les flexions supportées sont la conjugaison aux temps simples, et la déclinaison en genre et en nombre de toutes les catégories grammaticales. Concernant les accords, les syntagmes nominaux s'accordent en genre et en nombre, tandis que le syntagme verbal s'accorde avec son sujet que ce soit un nom propre ou commun, ou un pronom sujet.

JSrealB reste limité aux langues à flexion. L'outil reposant sur un système de règles appliquées de façon systématique, il ne permettrait pas de réaliser des textes dans une langue qui ne repose pas sur un tel fonctionnement.

Ensuite, la plus grande limitation est le lexique. En effet, pour être utilisé, un mot doit être dans le lexique. De même, les mots du lexique doivent être liés à une table de transformation pour leur flexion ou toute autre transformation comme l'élision.

5 Perspectives

JSreal est un outil permettant de réaliser notamment des phrases structurées en Sujet-Verbe-Objet (p. ex. *Il pleuvra demain midi.*), ou encore des phrases nominales (p. ex. *Chutes abondantes de neige cette nuit !*). Cette couverture minimaliste permet de réaliser des phrases pour le projet MeteoVis si ce n'est qu'il manque les modules `Date` et `Number` qui doivent être régionalisés pour réaliser les prévisions attendues. La date permet dans certains cas de déterminer le temps pour la conjugaison des verbes et le module `Number`, détermine le nombre grammatical pour l'accord des noms. Ces deux modules permettent donc d'adapter efficacement la réalisation au contexte. De plus, en permettant la régionalisation (p. ex. format de la date en fonction du lieu et de la langue), ils facilitent l'activité de traduction (p. ex. la date est définie dans un format standard dans la représentation syntaxique puis réalisée dans le format désiré). Le développement des modules `Number` et `Date` et l'intégration de JSrealB dans MeteoVis sont prévus avant la fin du PFE, c'est-à-dire, dans le mois suivant la soutenance. L'intégration consistera à définir les représentations syntaxiques des différentes prévisions pour permettre leur réalisation.

Au-delà, plusieurs cas ne sont pas supportés, nous pouvons notamment citer la négation qui fonctionne différemment dans les deux langues, généralement, en français deux adverbes *ne* et *pas* sont utilisés (p. ex. *il ne parle pas*) alors qu'en anglais un seul suffit *not* (p. ex. *he does not speak*), les contractions (p. ex. *can not* parfois contracté *cannot* en anglais ou *de le* devient *du* en français), l'élision (p. ex. *le homme* devient *l'homme*), ou encore la possession ne sont pas implémentées pour le moment. Ces règles nécessitent des informations à insérer dans les lexiques, mais surtout des mécanismes algorithmiques à implémenter dans la plus haute couche de JSrealB, au niveau des syntagmes. Enfin, de nombreuses caractéristiques des lexiques ne sont pas encore prises en compte (p. ex. pour l'anglais le fait qu'un nom soit dénombrable ou indénombrable) ce qui peut causer des erreurs lors de la réalisation (p. ex. `NP(D("a"), N("milk"))` se réalise en *a milk* alors qu'il devrait se réaliser en *the milk*).

Le projet demande donc à être poursuivi au-delà du PFE pour codifier de nouvelles règles, ainsi, nous pourrons réaliser davantage de types de phrases. Néanmoins, le cœur du programme a été développé et les différents mécanismes qui ont été mis en place permettant d'étendre rapidement la couverture.

6 Conclusion

Dans ce rapport, nous avons présenté l'outil JSrealB qui repose sur des règles algorithmiques pour propager les caractéristiques de la phrase, ou de ses éléments jusqu'aux éléments terminaux et un moteur d'application systématique de règles à partir de tables pour les transformations de ces éléments. Cette approche permet de réaliser des textes en anglais et en français, tout en limitant les développements spécifiques et d'ajouter de nouvelles langues à moindre coût, résolvant ainsi la problématique du multilinguisme des réalisateurs de textes. L'objectif est maintenant que JSreal atteigne un niveau de couverture de l'anglais et du français comparable à celui de SimpleNLG-EnFr.

Références

Bescherelle, *Bescherelle La conjugaison pour tous*, Hatier, 2012.

N. Daoust and G. Lapalme, *JSreal: A Text Realizer for Web Programming*, *Language Production, Cognition, and the Lexicon*, Springer, 2014, pp. 363-378.

B. Delaunay and N. Laurent, *Bescherelle La grammaire pour tous*, Hatier, France, 2013.

A. Gatt and E. Reiter, *SimpleNLG: A realisation engine for practical applications*, in A. f. C. Linguistics, ed., *12th European Workshop on Natural Language Generation*, Athens, Greece, 2009, pp. 90-93.

M. Grevisse, *Le bon usage, grammaire française, 11e édition*, Duculot, Louvain-la-Neuve, Belgique, 1980.

M. Mouine and G. Lapalme, *Présentation personnalisée des informations environnementales*, *Informatique et recherche opérationnelle*, Université de Montréal, 2014, pp. 129.

M. Mouine and G. Lapalme, *Using clustering to personalize visualization*, in IEEE, ed., *16th International Conference on Information Visualisation*, Montpellier, France, 2012, pp. 258-263.

E. Reiter and R. Dale, *Building natural language generation systems*, Cambridge University Press, 2000.

F. O. Ters, D. Reichenbach and G. Mayer, *L'échelle Dubois-Buyse d'orthographe usuelle française*, Messeiller, France, 1964.

P.-L. Vaudry and G. Lapalme, *Adapting SimpleNLG for bilingual English - French realisation*, in A. f. C. Linguistics, ed., *14th European Workshop on Natural Language Generation*, Sofia, Bulgaria, 2013, pp. 183-187.

R. Verret, D. Vigneux, J. Marcoux, F. Petrucci, C. Landry, L. Pelletier and G. Hardy, *Scribe 3.0, a product generator*, *13th International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography and Hydrology*, Long Beach, CA, United States, 1997, pp. 392-395.

Webographie

L. Bronner, *Des robots au « Monde » pendant les élections départementales ? Oui… et non*, Le Monde, 2015.

Environnement Canada, *Meteorological Product Exchanger*, 2015.

P. Shoebottom, *The differences between English and French*, in F. I. School, ed., *A guide to learning English*, Frankfurt International School, 1996.