Université de Montréal

JSreal : un réalisateur de texte pour la programmation web

Nicolas Daoust

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maitre ès sciences (M.Sc.)
en informatique

Septembre 2013

Université de Montréal
Faculté des arts et sciences

Ce mémoire, intitulé

JSreal : un réalisateur de texte pour la programmation web

présenté par

Nicolas Daoust

a été évalué par un jury composé de

Philippe Langlais
président

Guy Lapalme
directeur de recherche

Michel Boyer

Mémoire accepté le 17 octobre 2013

Résumé

La génération automatique de texte en langage naturel est une branche de l'intelligence artificielle qui étudie le développement de systèmes produisant des textes pour différentes applications, par exemple la description textuelle de jeux de données massifs ou l'automatisation de rédactions textuelles routinières. Un projet de génération de texte comporte plusieurs grandes étapes : la détermination du contenu à exprimer, son organisation en structures comme des paragraphes et des phrases et la production de chaînes de caractères pour un lecteur humain ; c'est la réalisation, à laquelle ce mémoire s'attaque.

Le web est une plateforme en constante croissance dont le contenu, de plus en plus dynamique, se prête souvent bien à l'automatisation par un réalisateur. Toutefois, les réalisateurs existants ne sont pas conçus en fonction du web et leur utilisation requiert beaucoup de connaissances, compliquant leur emploi.

Le présent mémoire de maîtrise présente JSreal, un réalisateur conçu spécifiquement pour le web et facile d'apprentissage et d'utilisation. JSreal permet de construire une variété d'expressions et de phrases en français, qui respectent les règles de grammaire et de syntaxe, d'y ajouter des balises HTML et de les intégrer facilement aux pages web.

Mots-clés : traitement automatique du langage naturel, génération automatique de texte, réalisation de texte

Abstract

Natural language generation, a part of artificial intelligence, studies the development of systems that produce text for different applications, for example the textual description of massive datasets or the automation of routine text redaction. Text generation projects consist of multiple steps : determining the content to be expressed, organising it in logical structures such as sentences and paragraphs, and producing human-readable character strings, a step usually called realisation, which this thesis takes on.

The web is constantly growing and its contents, getting progressively more dynamic, are well-suited to automation by a realiser. However, existing realisers are not designed with the web in mind and their operation requires much knowledge, complicating their use.

This master's thesis presents JSreal, a realiser designed specifically for the web and easy to learn and use. JSreal allows its user to build a variety of French expressions and sentences, to add HTML tags to them and to easily integrate them into web pages.

Keywords : natural language processing, natural language generation, text realisation

Table des matières

Liste des figures

1 Introduction

Les projets de génération de texte comportent plusieurs étapes : la détermination du contenu, son organisation, puis sa réalisation en langage naturel. Plusieurs réalisateurs sont déjà disponibles dans plusieurs langages de programmation et pour plusieurs langages naturels. En général, ils sont au coeur des applications qui les utilisent et mettent en oeuvre des modèles syntaxiques détaillés dont l'utilisation requiert beaucoup de connaissances théoriques.

JSreal

Le projet JSreal s'inscrit dans une perspective différente : la programmation web.

La création d'une page web basique est extrêmement simple : il suffit d'entrer des caractères dans un fichier texte d'extension `html` et de l'ouvrir dans un navigateur pour voir ces caractères affichés. Des balises HTML peuvent ensuite ajouter du formatage et d'autres fonctionnalités.

Plusieurs langages de programmation interagissent habilement avec le web, mais le plus simple et le plus fondamental est JavaScript, qui ne requiert aucune installation, peut être intégré à une page web en quelques lignes et est reconnu par tous les navigateurs. De plus, la syntaxe de JavaScript est parmi les plus faciles à apprendre.

JSreal approche donc la réalisation d'un angle nouveau : l'accessibilité. Sa couverture syntaxique est inférieure à celle d'autres réalisateurs, mais il est trivial à ajouter à une page web, s'apprend rapidement et utilise du code beaucoup plus concis que celui des autres réalisateurs.

Organisation du mémoire

Ce mémoire commence par un survol de l'historique de la génération de langage naturel et de la place que la réalisation de texte y occupe, terminant par la présentation de quelques réalisateurs.

JSreal est ensuite introduit et ses fonctionnalités sont énumérées, appuyées par quelques exemples. S'ensuit une démonstration d'utilisation de JSreal dans un problème complet de génération de texte.

Finalement, la conception de JSreal est décrite, mentionnant quelques décisions et résumant son fonctionnement interne.

Plusieurs figures du mémoire montrent des spécifications de JSreal ; la réalisation correspondante peut être indiquée comme commentaire dans le code ou servir de titre à la figure.

Site web

Pour que JSreal soit particulièrement accessible, toute l'information à son sujet se trouve sur son site web.[1] Comme la figure 1 le montre, la majorité des chapitres de ce mémoire y sont disponibles ; leur contenu est identique.

Le site web contient aussi la documentation exhaustive de JSreal,[2] énumérant chacune de ses fonctionnalités et indiquant comment s'en servir. Tout au long du contenu, certains mots-clés peuvent être cliqués pour faire afficher une définition.

FIGURE 1: Site web de JSreal

Une page de test[3] (figure 2) permet d'essayer en ligne des spécifications JSreal pour en voir la réalisation.

Au cours du développement de JSreal, j'ai mis en place des tests de régression (figure 3 pour éviter d'accumuler des problèmes en augmentant la couverture du programme. Ces tests sont tous disponibles en annexe,[4] commentés pour indiquer ce qu'ils vérifient.

Enfin, le site web inclut une liste exhaustive, en ordre alphabétique, des méthodes disponibles pour modifier les objets de JSreal ainsi que les valeurs qu'elles peuvent prendre.[5]

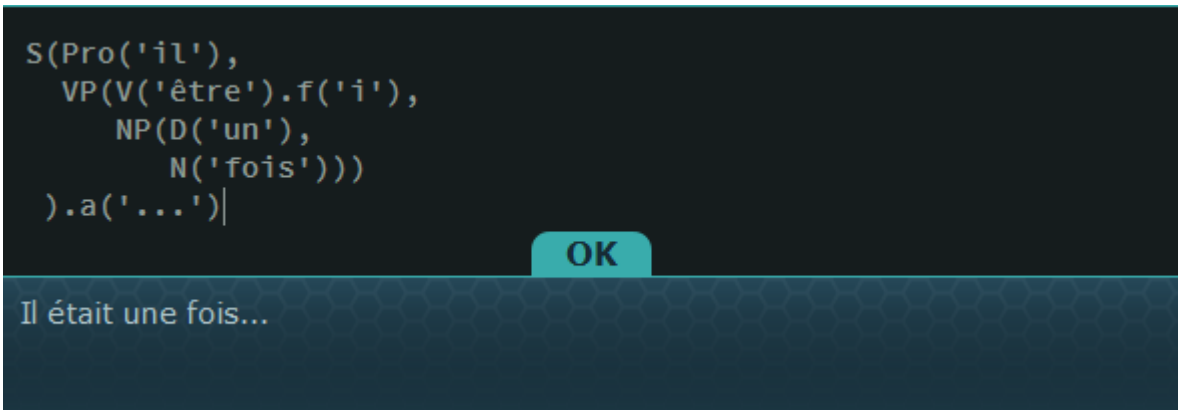

FIGURE 2: Page de test de JSreal

FIGURE 3: Tests de régression de JSreal

2 Contexte

En tant que réalisateur de texte, JSreal s'inscrit dans le domaine de la génération automatique de texte en langage naturel (GAT), un aspect du **traitement automatique du langage naturel** (TALN), une branche de l'intelligence artificielle.

Historique

La question du TALN a commencé à être étudiée en profondeur dès les débuts de l'informatique moderne. En 1948, Claude Shannon produisait des séries de mots (dépourvues de sens) à partir de probabilités markoviennes de transition entre eux ; plusieurs méthodes utilisées aujourd'hui découlent directement de ces expériences. Puis, en 1950, Alan Turing a publié son célèbre article "Computing Machinery and Intelligence", dans lequel il proposait qu'un ordinateur qui imiterait assez bien la conversation humaine pourrait être considéré, à toutes fins pratiques, intelligent ; ce "jeu d'imitation" est maintenant connu sous le nom de test de Turing.

C'est dans les années 60 que Joseph Weizenbaum a écrit un des premiers agents conversationnels, ELIZA, qui pouvait entretenir une conversation textuelle limitée avec un humain. Le plus mémorable des scripts d'ELIZA, DOCTOR, devait parodier un psychothérapeute ; ses répliques évasives, dont des questions reformulant la dernière phrase de l'interloctuteur, parvenaient parfois à convaincre celui-ci qu'il conversait avec une personne.

La majorité du travail en GAT de ces premières décennies s'inscrivait dans le cadre de la traduction automatique, mais par la suite, plusieurs travaux se sont penchés sur l'expression en langage naturel d'informations non disponibles sous cette forme. Ces premières incursions ont montré définitivement que la GAT présente des défis entièrement différents de ceux de la compréhension du langage naturel, en faisant un champ de recherche distinct.

2.1 Génération

La **génération automatique de texte en langage naturel** (GAT) a comme objectif d'exprimer en langue humaine du contenu qui n'existe pas, à la base, sous forme textuelle.

Application

En pratique, la GAT sert principalement à décrire textuellement des jeux de données massifs, exploitant la capacité des ordinateurs à les analyser plus rapidement que les humains. Même dans les cas où certains experts sont capables d'interpréter ces données par eux-mêmes, il est utile de pouvoir n'en présenter que des résumés ou d'autres aspects pertinents en langage naturel à un non-expert.

Au-delà des analyses rapides, un autre avantage des ordinateurs est leur capacité à effectuer des tâches routinières. Dans l'exercice de leur emploi, plusieurs personnes sont tenues d'exprimer à répétition de l'information dont seuls certains détails changent d'une fois à l'autre, par exemple des réponses aux

clients en service à la clientèle. La GAT peut automatiser une partie de cette rédaction, ne sollicitant l'humain que pour les détails changeant et lui sauvant un temps considérable.

Des systèmes de GAT ont déjà été employés dans divers domaines[6,8] :

- **WeatherReporter**[6] est une étude de concept qui produit à partir de données numériques des rapports météorologiques rétrospectifs mensuels, commentant qualitativement la température et l'humidité et faisant ressortir les conditions climatiques inhabituelles.
- **FoG**[9] part également de données climatologiques numériques, mais elles sont entrées dans le simulateur climatique d'un superordinateur pour prédire des conditions météorologiques. Utilisé par Environnement Canada, FoG produit ses bulletins en anglais et en français.
- **ModelExplainer**[10] décrit textuellement les relations entre les objets de n'importe quel logiciel programmé par objet.
- **Peba**[11] compare textuellement les caractéristiques des animaux de sa base de données.

Compréhension et génération

Dans ses débuts, le TALN traitait la compréhension et la génération de langage naturel presque simultanément. L'une semble en effet l'inverse de l'autre : la compréhension convertit le langage humain en représentations informatiques et la génération convertit des représentations informatiques en langage humain.

En pratique, toutefois, ces procédés diffèrent fondamentalement.[7] La difficulté de la compréhension vient des nombreuses interprétations possibles d'un texte, dont la désambiguïsation requiert des connaissances sur le monde ; l'enjeu de la génération est quant à lui d'analyser de l'information et de déterminer comment la présenter textuellement.

C'est dans les années 70 que ces spécificités de la génération se sont clarifiées, en faisant un champ d'étude distinct du reste du TALN. Dans les années 80, le développement de plusieurs systèmes de GAT a révélé que la génération consiste en plusieurs problèmes distincts qui peuvent regroupés en deux grandes catégories : la planification et la réalisation.[6]

2.2 Planification

La **planification** est l'étape de la génération de texte pendant laquelle l'information à communiquer est déterminée puis organisée. Les termes et exemples utilisés ici sont une traduction de Reiter et Dale.[6]

Détermination du contenu

Un système de génération de texte intervient dans des situations où l'information existe sous forme non textuelle ; cette information peut être quantitative, qualitative ou les deux. Plusieurs enjeux doivent être considérés pour déterminer le contenu à présenter :

- Les mêmes informations peuvent mener à du contenu plus ou moins détaillé ; par exemple, si WeatherReporter considérait une année à la fois plutôt que un mois, beaucoup de son contenu habituel devrait être omis.
- Il arrive que le contenu à exprimer ne soit pas explicitement disponible dans les données et qu'il résulte de calculs ou de simulations, comme avec FOG.
- Que l'éventuel lecteur soit ou non expert dans le domaine couvert affecte quel contenu doit lui être présenté — et comment.

Cette étape de la génération de texte dépend entièrement de son domaine d'application et son implantation doit donc y être spécifique. On pourra faire appel à un expert ou préparer un corpus d'exemples de contenu ; il n'existe pas de règles générales.

Occasionnellement, l'information qui doit être présentée a déjà été préparée par l'application à laquelle le générateur de texte est intégré, dans lequel cas ce dernier n'entre en jeu qu'à la prochaine étape.

Structuration du document

Une fois que le contenu a été déterminé, il doit être structuré. C'est à cette étape qu'est décidé, en particulier, l'ordre dans lequel l'information est présentée et comment elle est découpée en blocs logiques comme des paragraphes et en phrases, voire avec des titres, des listes ou autres. Dans le programme, cette étape est généralement combinée à celle de la lexicalisation.

Lexicalisation

La dernière étape de la planification est moins spécifique au domaine d'application et consiste à choisir les structures et mots exacts pour exprimer l'information, produisant une spécification de texte.

Les enjeux de la lexicalisation viennent principalement des nombreuses manières dont une information donnée peut être traduite en langage naturel. Par exemple, chacune des phrases suivantes contient la même information.

- La route sera fermée le 10 juin, la route sera fermée le 11 juin, la route sera fermée le 12 juin, la route sera fermée le 13 juin et la route sera fermée le 14 juin.
- La route sera fermée le 10 juin, le 11 juin, le 12 juin, le 13 juin et le 14 juin.
- La route sera fermée les 10, 11, 12, 13 et 14 juin.
- La route sera fermée du 10 au 14 juin.
- La route sera fermée 5 jours à partir du 10 juin.

Un lecteur trouverait la première phrase extrêmement redondante et la deuxième maladroite. Les trois autres seraient plus ou moins appropriées selon qu'il est nécessaire d'énumérer chaque jour ou que l'information la plus importante est la durée de la fermeture.

Ces exemples illustrent un problème important de la lexicalisation, soit celui de l'agrégation. Dans la première phrase, il est évident pour un lecteur que la même route est concernée par chacune des propositions ; dans la deuxième, les cinq jours appartiennent évidemment au même mois. L'agrégation

consiste à reconnaître ces répétitions pour les regrouper et s'applique de manière semblable dans tous les domaines et toutes les langues.

Une dernière considération est celle de la génération d'expressions référentielles. Une même entité peut, selon le contexte et les mentions précédentes, être désignée par des groupes nominaux plus ou moins détaillés ou être remplacée par un pronom. Par exemple, "Nicolas Daoust" sera nommé "Nicolas" ou "M. Daoust" selon le niveau de formalité et de simples "celui-ci" ou "lui" suffiront par la suite.

Quoique JSreal est un réalisateur, il peut aider à la lexicalisation :

- en insérant des mots simples comme des déterminants et des verbes auxiliaires ;
- en remplaçant sur demande des mots par le pronom approprié ;
- en modifiant des pronoms selon leur fonction ;
- en générant lui-même des expressions temporelles.

2.3 Réalisation

La **réalisation de texte** est l'étape de la génération de texte pendant laquelle l'information à transmettre, déjà bien structurée, est exprimée textuellement, dans une langue humaine. C'est à cette étape que les **réalisateurs** comme JSreal entrent en jeu.

Entrée

À la fin de l'étape de planification, un générateur de texte a produit une **spécification de texte**, une structure en arbre qui contient tout le détail de l'information à présenter. Cette structure doit être construite sur mesure pour le réalisateur de texte utilisé.

Par exemple, dans SimpleNLG-EnFr,[12,13] la spécification du syntagme nominal "les très belles maisons spacieuses" :

- crée le syntagme nominal à partir des mots "le" et "maison" ;
- crée un syntagme adjectival à partir de "beau" ;
- crée un syntagme adverbial pour "très" ;
- ajoute ce syntagme comme modificateur de celui de "beau" ;
- ajoute des modificateurs au syntagme nominal ;
- met le syntagme nominal au pluriel ;
- demande la réalisation finale sous forme de chaîne de caractères.

```
NPPhraseSpec theMan = 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);
String realisation = realiser.realise(greeting).getRealisation();
```

FIGURE 4: Spécification SimpleNLG-EnFr

Une spécification en JSreal peut souvent être créée en une seule expression. Puisque les noms des fonctions utilisées ressemblent à ceux des structures syntaxiques classiques, elle rappelle alors la structure syntaxique du groupe de mots qu'elle produit. Dans cet exemple, la méthode `n` met le syntagme nominal au pluriel et la méthode `d` lui assigne un déterminant défini. Chaque mot voit sa catégorie définie explicitement ; leur ordre d'entrée dans leurs groupes est sans importance.

```
NP(N('maison'),
   AP(A('beau'),
      Adv('très')),
   A('spacieux')
  ).d('d').n('p')
```

FIGURE 5: Spécification JSreal

Pour que le réalisateur choisi dispose de toutes les instructions qu'il requiert, il peut être nécessaire d'ajouter quelques opérations additionnelles à la fin de la planification.

Planification et réalisation

Alors que la planification organise l'information, la réalisation n'est pas spécifique à un domaine mais plutôt à une langue.

Un réalisateur de texte encode le lexique, la grammaire et la syntaxe d'une langue. Il s'applique indifféremment à tous les domaines à condition que ceux-ci emploient des mots qu'il connaît ou qu'il peut interpréter correctement. Ainsi, la planification et la réalisation peuvent être développées séparément — et le sont généralement.

Certains générateurs de texte se passent d'un réalisateur dédié et se contentent de quelques règles grammaticales programmées manuellement, mais chacune de ces règles pourrait être automatisée par un réalisateur complet, qu'il s'agisse de la déclinaison irrégulière d'un nom, de la position inhabituelle d'un adjectif ou de la formation d'une phrase interrogative.

Formatage

Au-delà des spécificités d'une langue, un réalisateur peut également prendre en charge le formatage du document quand sa spécification le demande. À la base, il peut commencer chaque phrase par une

majuscule et la terminer par un point.

Le reste du formatage, soit les titres, listes, paragraphes et autres éléments, dépend du système de présentation utilisé, comme HTML ou LaTeX. Un titre de section, dans ces systèmes, prendrait respectivement les formes `<h1>Titre</h1>` et `\section{Titre}`.

Comme JSreal est destiné à la présentation de pages web, il intègre exclusivement le HTML, dont le formatage peut être enrichi par l'utilisation de feuilles de style en cascade (CSS).

2.4 Autres réalisateurs

Beaucoup de langages de programmation ne disposent pas de réalisateurs complets, ce qui implique d'écrire un réalisateur de texte spécifique à chaque projet de génération. Un tel réalisateur comporte donc des patrons de phrases et de documents dont seuls certains aspects peuvent être modifiés et qui ne gèrent que quelques cas particuliers ; il requerrait beaucoup de modifications pour être adapté à un autre projet. Là où un réalisateur complet encode une grande diversité de mots, syntagmes et règles de grammaire, un réalisateur spécifique ne couvre que les quelques mots et structures qu'il doit pouvoir produire et les traite beaucoup plus naïvement.

Certains réalisateurs complets existants, dont KPML et RealPro, peuvent en théorie être utilisés dans la programmation web. Toutefois, deux aspects de ces réalisateurs compliquent cet emploi :

- aucun n'est écrit dans un langage natif au web, comme JavaScript, alors interagir avec eux requiert nécessairement des détours ;
- l'usage d'un réalisateur est généralement complexe, requérant beaucoup de connaissances théoriques, d'apprentissage et de consultation de références.

À cause de ces complications, l'étape de la réalisation d'un problème de génération sur le web est presque systématiquement mieux servie par un réalisateur spécifique à ce problème.

KPML

Pour utiliser un réalisateur de texte, un projet de génération de texte doit lui fournir une spécification de texte. En général, cette spécification prend la forme d'une longue chaîne de caractères que le réalisateur peut interpréter. Par exemple, pour générer la phrase "La température est basse.", le réalisateur KPML[14] demande la spécification suivante :

```
(PA / PROPERTY-ASCRIPTION
  :TENSE present
  :DOMAIN (WE / OBJECT :LEX température :DETERMINER le)
  :RANGE (B / QUALITY :LEX bas)
  :SET-NAME tempComment
)
```

FIGURE 6: Spécification KPML

Quoiqu'une telle spécification soit compréhensible, en produire une requiert beaucoup de connaissances sur KPML. Dans ce dernier exemple, le verbe "être" est sous-entendu par la procédé relationnel PROPERTY-ASCRIPTION, puis le sujet et attribut respectivement assignés par DOMAIN et RANGE.

RealPro

Un autre réalisateur, RealPro,[15,16] développé par CoGenTex, permet aussi une grande abstraction dans ses structures syntaxiques, pouvant automatiser des aspects comme la génération d'expressions référentielles. Le système de génération de texte ModelExplainer utilise RealPro comme réalisateur.

L'exemple suivant montre comment "be" est central à la structure syntaxique reliant "this" et "test".

```
DSYNTS:
BE [ ]
( I this [ class:demonstrative_pronoun number:sg ]
  II test [ class:common_noun article:indef ] )
END:
```

FIGURE 7: "This is a test."

Tel que mentionné précédemment, KPML et RealPro permettent l'ajout de balises HTML, RealPro par le trait `sgml` :

```
DSYNTS:
BE [ ]
( I this [ number:sg ]
  II CoGenTex [ class:proper_noun
                sgml:"a href=http://www.cogentex.com" ] )
END:
```

FIGURE 8: RealPro : ajout de balise HTML

SURGE

SURGE[17] insiste comme KPML sur les fonctions syntaxiques des composants de ses structures. Créer une phrase requiert d'en spécifier plusieurs caractéristiques, puis d'en indiquer les constituants à partir du "thème" de chacun, comme "agent", "procédé" et "possesseur". Ces thèmes sont ensuite assignés aux fonctions syntaxiques comme le sujet, le syntagme verbal et le complément d'objet direct. Produire une spécification pour SURGE requiert donc une bonne maîtrise de la théorie syntaxique.

Pour la défense de KPML, RealPro et SURGE, la complexité de leurs spécifications leur permet d'exploiter des connaissances grammaticales et syntaxiques de loin supérieures à celles de réalisateurs plus accessibles, comme SimpleNLG.

SimpleNLG

La majorité des réalisateurs complets existants sont considérés si compliqués que SimpleNLG,[18,19] un réalisateur en Java, indique dans son nom, et se définit, par sa simplicité.[20] Pour générer la phrase "La

température est basse", SimpleNLG-En-Fr (après chargement du lexique et quelques importations et déclarations) utilise le code suivant :

```
NPPhraseSpec theTemp = factory.createNounPhrase("le", "température");
AdjPhraseSpec adjLow = factory.createAdjectivePhrase("bas");
SPhraseSpec tempComment = factory.createClause(theTemp, "être", adjLow);
```

FIGURE 9: Spécification SimpleNLG-EnFr

Chaque ligne a été facile à créer, sans trop consulter de documentation ; le résultat est très lisible et indique clairement la structure finale qui est construite.

Contrairement à celles des autres réalisateurs, les spécifications de SimpleNLG ne sont pas que des longues chaînes de caractères écrites dans un langage spécifique au réalisateur et interprétées d'un coup. SimpleNLG est une bibliothèque Java et chaque composante de ses spécifications est un objet Java, permettant une intégration beaucoup plus facile à tout projet également écrit en Java.

JSreal

Parmi les réalisateurs décrits dans cette section, c'est de SimpleNLG que JSreal se rapproche le plus, étant lui aussi une bibliothèque intégrée à un autre programme en quelques lignes. Comme SimpleNLG, JSreal demande à l'usager qu'il groupe ses mots au lieu d'utiliser de la théorie syntaxique plus abstraite.

Tout utilisateur de SimpleNLG sera immédiatement familier avec JSreal. Pour produire "La température est basse.", une spécification JSreal presque identique à celle de SimpleNLG fonctionnerait :

```
var theTemp = NP(D('le'), N('température'));
var adjLow = A('bas');
var tempComment = S(theTemp, VP(V('être'), adjLow));
```

FIGURE 10: Spécification JSreal semblable à SimpleNLG

Une différence notable est que JSreal demande que la catégorie de chaque mot soit définie par l'usager ; avec SimpleNLG, "le", "température" et "être" étaient interprétés sans cette exigence. Et en pratique, la concision des spécifications de JSreal leur permet une écriture plus brève :

```
var tempComment = S(N('température').d('d'), VP(V('être'), A('bas')))
```

FIGURE 11: Spécification JSreal abrégée

En comparaison à KPML et RealPro, qui permettent l'intégration de balises HTML, JSreal comporte beaucoup plus d'options : il permet aussi de créer des balises autonomes, fournit des méthodes qui s'occupent pour l'usager de la "syntaxe" des attributs, automatise l'entrée de liens et de courriels et accepte les balises sans contenu (par exemple `img`). De plus, que JSreal soit conçu dans un langage de programmation web garantit son implantation facile dans une page web.

3 JSreal

JSreal est un réalisateur de texte écrit en JavaScript qui produit des expressions et phrases françaises bien formées. Comme tout réalisateur, il peut être utilisé seul, pour des démonstrations linguistiques, ou être intégré à des projets complexes de génération de texte. Mais JSreal est avant tout destiné aux programmeurs web, leur permettant tant d'accorder quelques mots sans soucis que de générer des documents HTML complexes.

JavaScript

Comme son nom le suggère, JSreal est écrit en JavaScript et s'utilise avec ce langage. Son usage dans une page web requiert d'abord le chargement du lexique `lex-fr.js`, puis du fichier de base `JSreal.js`. Les deux étant disponibles publiquement, il suffit par exemple d'insérer le code suivant à l'intérieur de la balise `head` de la page :

```
<script src="http://daou.st/JSreal/lex-fr.js"></script>
<script src="http://daou.st/JSreal/JSreal.js"></script>
```

FIGURE 12: Charger JSreal dans une page web

JSreal met au service de l'usager plusieurs fonctions JavaScript, chacune créant un objet correspondant à son nom de fonction. Par exemple, la fonction `N` crée un nom et `NP`, pour *noun phrase*, un syntagme nominal. Il est possible d'assigner ces objets à des variables.

```
var tb = AP(A('beau'), Adv('très'))
```

FIGURE 13: Assignation d'objet JSreal à une variable

Ces objets JavaScript peuvent être utilisés comme constituants l'un de l'autre ou être modifiés par des méthodes. Ils peuvent aussi être utilisés comme des chaînes de caractères, demandant automatiquement leur réalisation. Dans le code suivant, la chaîne `str` est formée de la concaténation de `"de "` et de l'objet `tbm`, qui est réalisé à ce moment, retournant la chaîne de caractères qui le représente ; `str` contient donc "de très belles maisons".

```
var tbm = NP(N('maison'), tb)
tbm.n('p')
var str = 'de '+tbm
```

FIGURE 14: Réalisation d'objet JSreal en chaîne de caractères

3.1 Fonctionnalités

Cette section présente plusieurs fonctionnalités de JSreal mais n'est pas exhaustive : la documentation web[2] les énumère toutes. Tous les exemples de spécifications peuvent être essayés en ligne.[3]

Mots

Pour créer un mot avec JSreal, il suffit d'en entrer le lemme, sa forme de base, dans la fonction correspondant à sa catégorie ; le mot "son", par exemple, est entré `D('son')` ou `N('son')` selon qu'il sert de déterminant ou de nom.

Un mot est automatiquement transformé en groupe de mots par certaines méthodes :

- `add`, qui lui ajoute un autre mot ;
- `d`, qui lui assigne un déterminant ;
- `p`, qui l'introduit par une préposition ;
- `sub`, qui l'introduit comme subordonnée par un mot-outil.

Morphologie

JSreal fait prendre à chaque mot la bonne forme, en particulier en genre et en nombre pour les noms, déterminants et adjectifs et en personne, en mode et en temps pour le verbe. Il peut donc servir de simple dictionnaire de déclinaison et de conjugaison :

```
N('garçon').g('f') // fille
N('oeil').n('p') // yeux
V('être').pe(3).n('p').f('i') // étaient
```

FIGURE 15: Déclinaison et conjugaison

Si un mot n'indique pas de forme irrégulière dans le lexique, JSreal utilise des règles détaillées pour déterminer la forme fléchie et peut donc modifier correctement des mots inconnus ou inventés.

```
N('Romulien').g('f').n('p') // Romuliennes
VP(V('texter'), Pro('moi')).f('ip') // texte-moi
```

FIGURE 16: Flexion de mots inconnus

Les règles du féminin et du pluriel, tirées du Bon Usage (Grevisse), sont exhaustives et modifient les mots selon leur terminaison. Les adjectifs qui terminent en "eur" le changent en "euse" s'ils correspondent à un verbe qui termine en "er" ou "ir", mais en "eure" sinon. Quant aux verbes, leurs règles de conjugaison ne sont présentement complètes que pour le premier groupe ; le lexique pallie à cette faiblesse pour les autres groupes.

Les pronoms peuvent changer de forme selon leur fonction dans un syntagme.

```
S(CP(Pro('tu'), Pro('je')), VP(V('rejoindre'), Pro('elle')))
```

FIGURE 17: "Toi et moi la rejoignons."

Certains mots font l'objet d'élision, de liaison ou de contraction si le mot qui les suit commence par une voyelle.

```
S(Pro('ce'),
  VP(V('être'),
     NP(D('ce'),
        N('ordinateur'))))
```

FIGURE 18: "C'est cet ordinateur."

```
J(S(N('homme').d('d'),
    VP(V('descendre'),
       N('singe').d('d').p('de'))),
  S(N('singe').d('d').pro(1),
    VP(V('descendre').ell(1),
       N('arbre').d('d').p('de')))
 ).c(',')
```

FIGURE 19: "L'homme descend du singe, lui de l'arbre."

Traits

Les traits d'un mot peuvent être tirés du lexique ou lui être attribués par son contexte ou par l'usager. Chaque trait est représenté par une ou quelques lettres et a une méthode qui lui correspond. Chacune de ces méthodes retourne l'objet sur lequel elle est appelée, permettant aux méthodes d'être enchaînées et appelées à l'intérieur d'autres groupes de mots.

```
NP(N('gars').g('f').n('p'), D('le'))
```

FIGURE 20: "les filles"

Lexique

Le lexique de JSreal est adapté de celui de SimpleNLG-EnFr et contient présentement environ 4 000 entrées ; la section Conception détaille le processus de conversion.

JSreal se réfère au lexique pour connaître, entre autres, le genre et le nombre d'un mot, ses formes irrégulières, si un adjectif vient avant le nom ou si un verbe est copule.

Il est possible pour l'usager d'ajouter au lexique des nouveaux mots ou d'ajouter des traits à des mots existants ; ici, `lex` est la variable contenant le lexique.

```
lex.add('sanglier', {fs:'laie'})
```

FIGURE 21: Ajout de "laie" comme féminin singulier irrégulier de "sanglier"

Groupe de mots

Les groupes de mots de JSreal permettent d'assembler plusieurs unités de manière grammaticalement cohérente. Comme pour les mots, chaque groupe est un objet JavaScript créé à partir d'une fonction dont le nom rappelle la catégorie, comme `NP` pour *noun phrase*.

Pour ajouter des constituants à un groupe de mots, il faut les lui passer comme arguments ou utiliser la méthode `add` :

```
S().t('exc').add(
  Pro('je'),
  V('travailler'))
```

FIGURE 22: "Je travaille !"

La catégorie de chaque constituant indique au groupe laquelle de ses fonctions il occupe. Cette fonction peut être spécifiée par l'usager grâce à la méthode `fct` ou en ajoutant le constituant avec une méthode `add` spécifique :

```
S(V('approcher')).addSubj(Adv('demain'))
```

FIGURE 23: "Demain approche."

Certains groupes de mots propagent des traits à leurs constituants, qu'ils aient été indiqués par l'usager, le contexte ou un desdits constituants. Par exemple, dans une proposition, le sujet transmet des traits au prédicat qui, lui, les propage à son verbe et à son attribut.

```
S(Pro('elle'), VP(V('être'), A('merveilleux')))
```

FIGURE 24: "Elle est merveilleuse."

Une subordonnée peut même propager des traits du nom ou pronom auquel elle est subordonnée jusqu'à un éventuel attribut de son verbe.

```
Pro('celle').add(
  S(Pro('tu'),
    V('choisir').f('pc')
   ).sub('que'))
```

FIGURE 25: "celle que tu as choisie"

Remarquez qu'ajouter "de prendre" au verbe aurait correctement annulé cet accord.

Coordination

La coordination (`CP`) et la juxtaposition (`J`) unissent plusieurs constituants par un coordonnant, respectant les règles quand à sa répétition et à l'emploi des virgules. Elles utilisent des règles détaillées pour déterminer leur genre, nombre et personne.

```
S(CP(Pro('elle'),
     Pro('tu'),
     Pro('je')),
  VP(V('aller'),
     Pro('y'))
```

FIGURE 26: "Elle, toi et moi y allons."

Elles peuvent aussi utiliser un symbole comme coordonnant et éliminer les espaces entre les mots.

```
NP(D('le'), J(N('vaisseau'), N('mère')).c('-')).n('p')
```

FIGURE 27: "les vaisseaux-mères"

De plus, quand elles unissent des propositions, elles ne mettent une majuscule qu'à la première et de la ponctuation qu'à la dernière.

```
J(S(Pro('je'),
    V('venir')),
  S(Pro('je'),
    V('voir')),
  S(Pro('je'),
    V('vaincre'))
 ).c(',').f('pc')
```

FIGURE 28: "Je suis venu, j'ai vu, j'ai vaincu."

Expressions temporelles

Exprimer une date ou une heure en langage naturel, en tenant compte des éléments redondants et des jours de la semaine, est une opération que JSreal peut automatiser grâce à la fonction `DT`, dont le temps peut être spécifié de plusieurs manières :

- par un objet `Date` JavaScript ;
- par une chaîne de caractères de format spécifique[22] ;
- avec des méthodes comme `y` (*year*), `d` (*day*) et `min`.

```
DT().y(2012).m(10).d(21).h(16).min(29)
```

FIGURE 29: "le dimanche 21 octobre 2012 à 16 h 29"

En combinant des `DT` avec la fonction `DTR` (*date and time range*) et le ton naturel (utilisé par défaut), il est possible de générer des expressions complexes comme les suivantes :

```
DTR(DT(new Date().getTime()-24*60*60*1000),
    DT(new Date().getTime()+24*60*60*1000)
   ).only('date')
```

FIGURE 30: "d'hier à demain"

```
DTR(DT().y(2012).m(10).d(14),
    DT().y(2012).m(10).d(17)
   ).noDay(true)
```

FIGURE 31: "du 14 au 17 octobre 2012"

Dans ce dernier cas, "octobre" et "2012" n'ont pas été répétés inutilement ; aurions-nous été en 2012 que toute mention de l'année aurait également été omise.

Nombres

Un nombre JavaScript peut être déclaré comme nom, adjectif ou déterminant. Il peut être réalisé :

- avec son apparence JavaScript : "12345.67" ;
- en écriture décimale française : "12 345,67" ;
- en toutes lettres : "douze mille trois cent quarante-cinq et soixante-sept centièmes" ;
- comme ordinal : "onzième".

Un nombre utilisé comme déterminant d'un nom lui transmet s'il est singulier ou pluriel ; un nombre déterminant ou adjectif s'accorde avec son nom. Quand un nombre est exprimé en toutes lettres, il peut respecter, au choix, l'ancienne grammaire ou la nouvelle, celle-ci insérant des tirets dans la majorité des espaces.

Paramètres

L'usager a accès à des paramètres, dont :

- si les chevrons des balises HTML sont remplacés par les entités correspondantes, pour éviter d'être interprétés par le navigateur ;
- le coordonnant par défaut de la coordination et de la juxtaposition ;
- le format par défaut des nombres et des expressions temporelles.

Linéarisation

JSreal comprend des règles syntaxiques qui lui indiquent comment assembler correctement les constituants de ses groupes. Ces règles impliquent tant l'organisation des différentes parties d'un syntagme que la place inhabituelle de certains mots.

```
NP(A('beau'),
   N('camion'),
   A('rouge'),
   D('le'),
   A('gros'))
```

FIGURE 32: Syntagme nominal aux constituants désordonnés

Dans la structure qui précède, un syntagme nominal (`NP`) est formé de cinq mots dans le désordre. Assembler ces mots naïvement produirait le charabia "beau camion rouge le gros". Toutefois, JSreal, aidé de son lexique, a des connaissances qui lui permettent de bien ordonner ce groupe de mots.

- Dans un syntagme nominal, le déterminant vient d'abord, suivi des adjectifs antéposés, puis du nom, puis des adjectifs postposés.
- Selon le lexique, les adjectifs "beau" et "gros" sont antéposés.

JSreal produira donc la réalisation correcte, "le beau gros camion rouge".

Formatage

JSreal peut intégrer aux textes qu'il produit n'importe quelle balise HTML et n'importe quel attribut de balise. Certaines balises particulièrement utilisées, comme `a` et `p`, disposent de leurs propres constructeurs, de même que certains attributs leurs propres méthodes.

Il est possible d'entremêler des balises HTML aux autres objets JSreal ; elles n'empêchent pas les traits grammaticaux de se propager normalement dans les syntagmes. Dans l'exemple suivant, "beau" est antéposé à "voiture" et s'y accorde malgré les balises qui les séparent.

```
NP(D('le'),
   N('voiture').tag('strong'),
   Em(A('beau')))
```

FIGURE 33: Traits malgré balises (spécification)

la *belle* **voiture**

FIGURE 34: Traits malgré balises (réalisation)

Il est préférable de définir les styles ailleurs, comme dans une feuille de style en cascade (CSS), et de spécifier seulement les identifiants et les classes avec JSreal. Dans le prochain exemple, `divContent`, `divID` et `divClass` sont des chaînes définies au préalable.

```
Div(divContent).id(divID).class(divClass)
```

FIGURE 35: Identifiant et classe

JSreal offre deux fonctions pour simplifier l'écriture de liens HTML. Dans les deux cas, il ajoute la balise `a` et recopie le contenu dans l'attribut `href` ; il précède le premier de `http://` si nécessaire et le deuxième de `mailto:`.

```
HL('JSreal.js')
```

FIGURE 36: Automatisation d'hyperlien

```
Email('n@JSreal.js')
```

FIGURE 37: Automatisation de courriel

Au final, JSreal peut générer le contenu complet d'une page web, mais il se limitera généralement au contenu du `body` ou d'une ou plusieurs `div`.

```
Div(N('propos').p('à').tag('h2'),
    Img().class('inline').src('JSreal.png'),
    P(CP(S(N('nom').d('p1'),
           VP(V('être'),
              N('Nicolas Daoust').tag('b'))),
         S(Pro('je'),
           VP(V('être'),
              N('créateur').d('d').add(
                N('JSreal').tag('a')
                  .href('http://daou.st/JSreal').p('de'))))),
      S(Pro('vous'),
        VP(V('pouvoir'),
           VP(V('contacter'),
              Pro('moi'),
              Email('n@JSreal.js').d('d').p('à'))))))
```

FIGURE 38: Contenu web complexe (spécification)

À propos

Mon nom est **Nicolas Daoust** et je suis le créateur de JSreal. Vous pouvez me contacter au n@daou.st.

FIGURE 39: Contenu web complexe (réalisation)

3.2 Exemples

JSreal met donc à la disposition de l'usager plusieurs outils :

- lexique
- règles de flexion

- règles syntaxiques
- expressions temporelles
- traitement des nombres
- génération de HTML

Les exemples simples qui suivent montrent quelques manières d'utiliser ces outils ; ils sont suivis d'une démonstration complète de génération de texte avec JSreal.

Usage direct du lexique

À la base, on peut accéder à l'information du lexique en passant par JSreal ; par exemple, le mot "vieux" peut révéler qu'il est antéposé, qu'il devient "vieil" devant une voyelle et que son féminin singulier "vieille" est irrégulier.

```
var v = A('vieux')
v.pos() // pre
v.l()   // vieil
v.fs()  // vieille
```

FIGURE 40: Traits tirés du lexique

En connaissant la structure JSON du lexique, il est possible de l'utiliser directement, par exemple pour produire une liste d'adjectifs antéposés ou de mots qui changent de forme devant une voyelle.

Tableau de conjugaison

En plus de l'information contenue dans son lexique, JSreal connaît les règles de déclinaison et conjugaison. Avec ces flexions et les outils de présentation, il est facile, par exemple, de créer un tableau de conjugaison. Soit `v` le lemme du verbe, `f` sa forme et `pe` sa personne :

```
var v = 'avoir', f = 'p'
var tb = Table()
for (var pe = 1; pe <= 6; pe++) {
  tb.add(TR(TD(
    S(Pro().pe(pe),
      V(v).f(f)
    ).cap(0).a('')
  )))
}
```

FIGURE 41: Génération d'un tableau de conjugaison

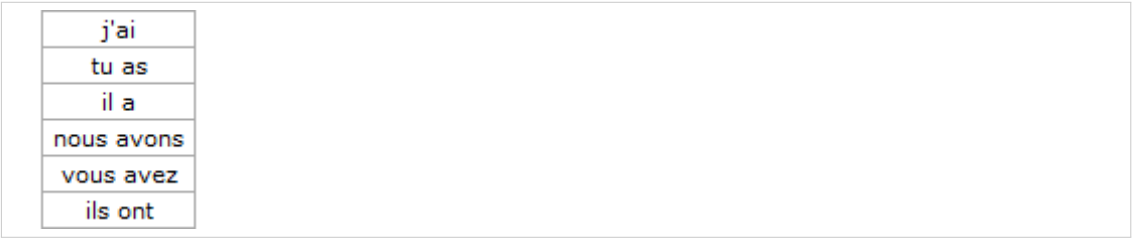

FIGURE 42: Tableau généré (verbe avoir, indicatif présent)

Il aurait été plus grammaticalement correct de spécifier la personne comme allant de 1 à 3 au singulier et au pluriel, mais JSreal accepte aussi les personnes 4 à 6 comme celles du pluriel. Les méthodes `cap` et `a` annulent respectivement la majuscule et le point qu'une proposition a lorsque indépendante.

Dénombrement

Il arrive qu'un programmeur veuille simplement exprimer une certaine quantité de quelque chose, par exemple d'un "item", et qu'il doive considérer manuellement les cas avec aucun item, un seul item ou plusieurs. Soit `nb` le nombre d'items et `nbitem` la chaîne finale :

```
var nbitem
if (nb == 0) nbitem = "aucun item"
else if (nb == 1) nbitem = "un item"
else nbitem = nb + " items"
```

FIGURE 43: Dénombrement d'items en JavaScript

Ce code n'est en rien complexe, mais au-delà des expressions complexes, une des forces de JSreal est de réduire la quantité de code requise dans des situations simples :

```
var nbitem = N('item').d(nb).num('l')
```

FIGURE 44: Dénombrement d'items avec JSreal

JSreal crée le nom "item", lui ajoute comme déterminant `nb` et demande la forme en toutes lettres du nombre. Cette construction produira donc des expressions comme "aucun item", "un item" et "dix-huit items". L'usager aurait pu modifier au préalable, par la ligne suivante, le paramètre qui indique à JSreal la forme par défaut des nombres :

```
JSreal.param.num = 'l'
```

FIGURE 45: Affectation de paramètre

4 Démonstration

Dans cette démonstration d'utilisation de JSreal, je maintiens sur une page web une liste en langage naturel des événements à venir que moi, Alice et Robert offrons à notre public. Comme l'information présentée utilise toujours des structures syntaxiques semblables, je veux la faire générer automatiquement pour obtenir un résultat comme le suivant :

Atelier à Laval

Le mercredi 25 septembre 2013 à 19 h, Alice sera à Laval pour le nouvel atelier *Exercices de réalisation*. Pour réserver, contactez-la au 555-2543.

Consultations à Montréal

Le vendredi 27 septembre 2013, Robert sera à Montréal pour des consultations de une demi-heure. Pour réserver, contactez-le au rob@JSreal.js.

Séjour à Granby

Du lundi 30 septembre au mercredi 2 octobre 2013, Alice et Robert seront à Granby, au 901 rue Principale, pour des consultations privées et deux ateliers.

- 30 septembre à 13 h: atelier classique *Principes de réalisation* avec Alice
- 2 octobre à 13 h: nouvel atelier *Exercices de réalisation* avec Robert

Pour réserver, contactez Robert au rob@JSreal.js.

Séjour à Longueuil

Les mercredi 2 et jeudi 3 octobre 2013, je serai à Longueuil pour plusieurs événements.

- 2 octobre à 19 h: nouvelle conférence *Pourquoi la réalisation?*
- 3 octobre à 13 h: nouvel atelier *Planification et réalisation*

Pour réserver, contactez-moi au 555-6426 ou au nic@JSreal.js.

FIGURE 46: Liste d'événements sur le web

L'exemple est fictif, mais fonctionnel. Il est inspiré d'une vraie situation de génération de texte[23] dont je me suis occupé dans le cadre d'un emploi ; elle a été modifiée pour démontrer plus de fonctionnalités de JSreal.

Il serait possible de produire la liste d'événements sans utiliser de réalisateur ; c'est l'approche que j'avais adoptée dans mon emploi. J'avais créé plusieurs chaînes de caractères fixes que j'assemblais selon les détails de chaque événement, en plus de fignoler des listes, des intervalles de dates et des détails comme les espaces, les majuscules, la ponctuation et le HTML. JSreal aurait simplifié plusieurs de ces opérations.

4.1 Données

Alice, Robert et moi disposons déjà d'un jeu de données qui rassemble toute l'information pertinente au sujet de nos événements ; nous voulons que le programme génère à partir de ces données du contenu web en langage naturel mis en page à notre goût.

Pour obtenir notre jeu de données, nous avons entré les détails de nos événements dans une application web simple à laquelle nous seuls avons accès et qui a produit une structure JSON comme la suivante :

```
var evList = [
{ date:'2013-09-25', ville:'Laval',cat:'at', h:'19:00', attr:'nouveau',
tit:'Exercices de réalisation', part:'a', res:'a' } ,
{ date:'2013-09-27', ville:'Montréal',cat:'cs', attr:'de une demi-heure',
part:'r', res:'r' } ,
{ date:'2013-09-30', ville:'Granby', adr:'au 901 rue Principale',cat:'cs',
attr:'privé', res:'r' } ,
{ date:'2013-09-30', ville:'Granby',cat:'at', h:'13:00', attr:'classique',
tit:'Principes de réalisation', part:'a' } ,
{ date:'2013-10-02', ville:'Granby',cat:'at', h:'13:00', attr:'nouveau',
tit:'Exercices de réalisation', part:'r' } ,
{ date:'2013-10-02', ville:'Longueuil',cat:'cf', h:'19:00', attr:'nouveau',
tit:'Pourquoi la réalisation?', part:'n', res:'n' } ,
{ date:'2013-10-03', ville:'Longueuil',cat:'at', h:'13:00',
tit:'Planification et réalisation', part:'n' }
]
```

FIGURE 47: Données d'événements en JSON

Les informations sur un événement sont les suivantes (certaines sont optionnelles) :

- `date`
- `ville`
- `adr` : adresse, si disponible
- `cat` : catégorie d'événement
  - `cs` : consultations
  - `at` : atelier
  - `cf` : conférence
- `h` : heure
- `attr` : adjectif ou groupe de mots qui précise l'événement
- `tit` : titre de l'événement
- `part` : chaîne de caractères qui inclut l'initiale de chaque participant
  - `a` : Alice
  - `r` : Robert
  - `n` : Nicolas (moi)
- `res` : initiale de la personne à contacter pour réservation

Quand plusieurs événements ont lieu dans la même ville pendant plusieurs jours consécutifs, ils constituent un seul "séjour" et doivent être regroupés. Si des consultations sont offertes, elles le sont pour toute la durée d'un tel séjour ; de manière semblable, un seul d'entre nous s'occupe des réservations d'un séjour donné et son identité n'est donnée que dans un des événements.

4.2 Programme

L'étape de détermination du contenu est déjà couverte par les données en JSON et comme JSeal s'occupe de la réalisation, il reste à coder la structuration du document et la lexicalisation.

Les extraits de code qui suivent sont tirés d'un programme complet[24] qui effectue la génération prévue.

Ajouts au lexique

Après vérification du lexique, je constate qu'il lui manque les mots "Alice", "Robert" et "consultation", alors je les y ajoute.

```
lex.add("Alice", { c:'N', i:1, g:'f' })
lex.add("Robert", { c:'N', i:1, g:'m' })
lex.add("consultation", { c:'N', g:'f' })
```

FIGURE 48: Ajouts au lexique

Autres informations

Comme Alice, Robert et moi ne sommes représentés dans les données que par nos initiales et que les identifiants d'événements ne sont eux-mêmes que deux lettres pour chacun, j'ajoute ces informations dans des structures JSON. J'en profite pour entrer la version JSreal de chaque mot : je suis personnellement représenté par le pronom "moi" et les consultations sont toujours multiples, donc au pluriel. J'inclus aussi les manières de contacter chacun d'entre nous.

```
var partInfo = {
  a: { word: N('Alice'), tel: 5552543, email: false },
  r: { word: N('Robert'), tel: false, email: 'rob@JSreal.js' },
  n: { word: Pro('moi'), tel: 5556426, email: 'nic@JSreal.js' }
}
```

FIGURE 49: Informations sur les particpants

```
var catWord = {
  cs: N('consultation').n('p'),
  at: N('atelier'),
  cf: N('conférence'),
  sj: N('séjour')
}
```

FIGURE 50: Informations sur les catégories d'événements

Contact

Comme chaque participant a toujours les mêmes manières d'être contacté, je fais automatiquement ajouter à `partInfo` la description de ces manières en JSreal.

- Le trait `num` ajoute au numéro de téléphone des tirets aux bons endroits.
- La fonction `Email` assigne le bon lien à l'adresse courriel.

```
for (p in partInfo) {
  var cdet = CP().c('ou')
  var tel = partInfo[p].tel
  if (tel)
    cdet.add(PP(P('à'), N(tel).num('t').d('d')))
  var em = partInfo[p].email
  if (em)
    cdet.add(PP(P('à'), Email(N(em)).d('d')))
  partInfo[p].cdet = cdet
}
```

FIGURE 51: Contact : détails

Traversée de la liste

Je traverse la liste, un événement à la fois, notant pour chaque "séjour" :

- sa ville et son adresse ;
- s'il inclut des consultations ;
- qui sera à au moins un de ses événements ;
- qui contacter pour réservation.

J'accumule aussi une liste des entrées JSON de chaque événement hormis les consultations. À la fin de chaque séjour, je génère tout le texte qui y correspond avant de passer au suivant.

Titre

Le titre d'un bloc d'événements varie s'il s'agit uniquement de consultations, uniquement d'un événement ou encore d'un mélange d'événements. Je note dans `cat` un de `cs`, `at`, `cf` ou `sj` selon le cas, puis :

```
var titre = H4(NP(cat, N(ville).p('à')))
// ex: <h4>Atelier à Laval</h4>
```

FIGURE 52: Titre de bloc d'événements

Participants

La variable `part` est un objet qui contient chaque initiale de participant, chacune assortie de la valeur `true` ou `false` selon leur présence ; je traduis cette liste en JSreal et la note dans `subj`.

```
var subj = CP(C('et'))
for (p in part)
  if (part[p]) subj.add(partInfo[p].word)
// ex: Alice, Robert et moi
```

FIGURE 53: Participants

Ville et adresse

Comme JSreal ne gère pas directement l'insertion, en cas d'adresse, j'ajoute une virgule manuellement après (méthode `a`) le nom de la ville et après l'adresse. Les variables `place` et `place2` seront ajoutées à une phrase plus tard.

```
var place = N(ville).p('à')
var place2 = false
if (adr) {
  place.a(',')
  place2 = PP(adr).a(',')
}
```

FIGURE 54: Ville et adresse

Raisons de la visite

Juste dans les quelques exemples-cibles que j'ai préparés, j'ai les expressions suivantes :

- "pour le nouvel atelier *Exercices de réalisation*"
- "pour des consultations d'une demi-heure"
- "pour des consultations privées et deux ateliers"
- "pour plusieurs événements"

Je commence par déclarer `rais`, qui contiendra les raisons de la visite. Les consultations sont détaillées ici plutôt que dans une éventuelle liste d'événements. Si elles ont un attribut, il est considéré adjectif s'il ne contient pas d'espace et syntagme prépositionnel sinon ; les données tiennent compte de cette distinction. Rappelons que `catWord.cs` contient `N('consultation').n('p')`.

```
var rais = CP(C('et'))
if (cs) {
  var attr = cs.attr.indexOf(' ') == -1? A(cs.attr) : PP(cs.attr)
  rais.add(catWord.cs.d('i').add(attr))
}
```

FIGURE 55: Raisons de la visite

S'il n'y a qu'un événement (outre les consultations), il est immédiatement détaillé et ajouté à `rais`. S'il y a une variété d'événements :

```
rais.add(NP(D('plusieurs'), N('événement')))
```

FIGURE 56: Raisons de la visite : plusieurs événements

S'il y a plusieurs événements, mais tous de la même catégorie, celle-ci est notée dans `cat` et la quantité dans `count` ; le trait `num` est utilisé pour faire afficher le nombre en toutes lettres.

```
rais.add(catWord[cat].d(count).num('l'))
```

FIGURE 57: Quantité d'un événement

Finalement, les raisons de la visite sont entrées dans le syntagme prépositionnel `pour`.

```
var pour = PP(P('pour'), rais)
```

FIGURE 58: Raisons de la visite : syntagme final

Date

Les dates de début et de fin d'un séjour sont déjà notées comme chaînes dans `date1` et `date2` ; si un séjour ne dure qu'une journée, `date2` est `false`.

- Si j'ai deux dates, je les entre dans un intervalle `DTR`, qui automatisera leur écriture.
- Sinon, j'ajoute l'heure de l'unique événement, si applicable ; il est entièrement décrit en tant que raison (dans `pour`) du séjour. Pour simplifier le code, considérons que la variable `h` contient déjà cette heure.

```
var date
if (date2)
  date = DTR(DT(date1), DT(date2))
else {
  if (h) date1 += 'T' + h
  date = DT(date1)
}
date.pos('beg')
```

FIGURE 59: Date de séjour, heure possible

Que la position (`pos`) de la date soit spécifiée comme au début (`beg`) en fait automatiquement un complément (à cette position) si elle fait partie d'une phrase.

Si un `DT` est réalisé à une date qui s'en rapproche suffisamment, certaines parties sont omises ou des expressions naturelles viennent remplacer les jours, mois et années. Pour clarifier : le séjour à Longueuil a normalement lieu "les mercredi 1 et jeudi 2 octobre 2013", mais si un membre du public le lit :

- le 25 septembre 2013, "de mercredi prochain au jeudi 3 octobre" s'affichera ;
- le 30 septembre 2013, "d'après-demain à ce jeudi" s'affichera.

Première phrase

Avec la date (`date`), le sujet (`subj`), le lieu (`place` et `place2`) et la raison (`pour`), je peux compléter la première phrase du bloc. Si `place2` (l'adresse) n'existe pas, son ajout n'a aucun effet.

```
var phr1 = S(date, subj, VP(V('être').f('f'), place, place2, pour))
```

FIGURE 60: Bloc d'événements : première phrase

Deux ajustements peuvent survenir automatiquement lors de la réalisation de cette phrase :

- si le sujet est uniquement le pronom "moi", il sera changé en "je" pour respecter sa fonction dans la phrase ;
- le verbe s'accordera avec la personne (1 ou 3) et le nombre (singulier ou pluriel) de son sujet.

Détails d'un événement

Les détails de chaque événement doivent être présentés quelque part, que ce soit directement avec les raisons de la viste ou dans une liste séparée. Pour en faire une liste, je commence par la déclarer comme une balise HTML `ul`, qui assignera automatiquement la balise `li` à chacun de ses constituants.

```
var listev = UL()
```

FIGURE 61: Création de liste non-ordonnée

Chaque événement qui doit être décrit est temporairement noté dans la variable `ev`. J'ai aussi déclaré au préalable la variable `item`, qui contiendra toute l'entrée d'un événement, et `det`, qui n'en contient que les détails non temporels. J'entre dans `item` la date et l'heure, indiquant au passage que le déterminant, le jour de la semaine et l'année sont superflus :

```
item = J().c(':')
item.add(DT(ev.date+'T'+ev.h).noDet(1).noDay(1).noYear(1))
// ex: 30 septembre à 13 h
```

FIGURE 62: Création d'item

J'entre d'abord dans `det` le nom de l'événement, tiré de `catWord`, et ses attributs, si applicable. Si l'événement a un titre, j'en fais un nom invariable entouré de guillemets et l'ajoute comme modificateur.

```
det = NP(catWord[ev.cat])
var attr = false
if (ev.attr) {
  attr = ev.attr.indexOf(' ') == -1? A(ev.attr) : PP(ev.attr)
  det.add(attr)
}
if (ev.tit) det.addMod(N(ev.tit).i(1).en('"'))
```

FIGURE 63: Détails d'événement

Comme JSreal connaît le genre et le nombre de l'événement, il accorde correctement tout adjectif qui lui est attribué. De plus, le lexique indique si un adjectif, comme "nouveau", prend position avant le nom et si celui-ci change de forme devant une voyelle. On peut donc obtenir "nouvelle conférence" et "nouvel atelier".

Dans les cas où les participants à un événement sont exactements les mêmes que ceux présents à tout le séjour, je ne les répète pas dans les détails. S'ils en diffèrent, par contre, je les ajoute ; je les coordonne d'abord, puis les combine au syntagme nominal existant dans un nouveau syntagme nominal pour m'assurer qu'ils viennent après les modificateurs déjà ajoutés. Ensuite, il ne reste qu'à ajouter `det` à `item`, puis `item` à `listev`.

```
avec = CP()
for (var k = 0; k < ev.part.length; k++) {
  var p = ev.part[k]
  avec.add(partInfo[p].word)
}
det = NP(det, PP(P('avec'), avec))
// ex: atelier classique avec Robert

item.add(det)
listev.add(item)
```

FIGURE 64: Finalisation d'item, ajout à liste

Réservation

La variable `ccon` est un syntagme verbal contenant la verbe "contacter" à l'impératif (`ip`). Ici, la personne (`pe`) est 5, un raccourci pour la deuxième personne du pluriel.

```
var ccon = VP(V('contacter').pe(5).f('ip'))
```

FIGURE 65: Contact : instruction

L'initiale de la personne désignée comme prenant les réservations est notée dans `res`. La variable `same` indique si le séjour a un seul participant et que celui-ci s'occupe des réservations ; son nom sera alors pronominalisé (`pro`), menant à des expressions comme "contactez-la". Ces variables sont utilisées pour ajouter l'information à `ccon`. La méthode `clone` copie le nom du participant pour éviter qu'il ne soit pronominalisé en permanence.

```
ccon.add(partInfo[res].word.clone().pro(same), partInfo[res].cdet)
```

FIGURE 66: Pronominalisation possible

La variable `pres` se fait assigner la phrase finale.

```
pres = S(ccon).addComp(PP('pour réserver'))
```

FIGURE 67: Bloc d'événements : deuxième phrase

Complétion

Je dispose maintenant de toutes les composantes pour décrire un bloc d'événements. S'il il a une liste d'événements, elle divise le bloc en deux paragraphes ; sinon, les deux phrases sont réunies en un seul paragraphe.

Je pourrais accumuler les séjours textuellement dans une longue chaîne de caractères `wholeText` créée au préalable ; les objets JSreal se réalisent automatiquement en chaînes quand ils sont traités comme tels. J'utiliserais ensuite `wholeText` comme bon me semble.

```
wholeText += titre
if (listev)
  wholeText += P(phr1) + listev + P(pres)
else
  wholeText += P(phr1, pres)
```

FIGURE 68: Liste d'événements : longue chaîne de caractères

Une autre possibilité serait que je dispose d'une division `evDiv` sur une page web et que j'y ajoute graduellement les descriptions des séjours. La méthode `node` retourne de mes composantes des éléments DOM, sur lesquels je peux utiliser `append`.

```
evDiv.append(titre.node())
if (listev) {
  evDiv.append(P(phr1).node())
  evDiv.append(listev.node())
  evDiv.append(P(pres).node())
} else
  evDiv.append(P(phr1, pres).node())
```

FIGURE 69: Liste d'événements : ajout graduel au DOM

4.3 Discussion

Tel que mentionné au début de la démonstration, il aurait été possible de produire une telle liste d'événements sans passer par JSreal, mais son utilisation simplifie beaucoup d'aspects.

Le genre et le nombre des participants et des événements entraîne l'accord automatique d'autres mots. Plusieurs événements sont déjà connus du lexique, mais le mot "consultation", de même que les noms propres, y sont facilement ajoutés.

Plusieurs listes sont créées sans savoir à l'avance combien d'éléments elles contiendront. Si elles n'ont pas d'éléments, elles sont simplement sautées, n'influençant pas la réalisation ; si elles en comportent

plusieurs, elles les séparent par les bons espaces, ponctuation et balises. De manière semblable, d'autres groupes ont un nombre d'unités variable, mais leur ajout est toujours simple.

Les dates et heures sont indiquées de manière numérique dans les données ; elles sont entrées dans JSreal en quelques lignes et produisent des expressions naturelles qui tiennent en compte beaucoup de détails. Il aurait été possible d'accepter comme données d'autres formats de date ou heure avec peu de modifications au programme.

Faire de tout ce texte un document web ne requiert que de l'encadrer dans des balises HTML grâce aux outils offerts par JSreal, qui sont même utilisés au milieu du texte pour mettre les titres en italiques et faciliter l'entrée des adresses courriels.

Finalement, à aucun moment les majuscules, la ponctuation et les espaces n'ont à être gérés. Les titres et phrases commencent toutes par une majuscule et les phrases et paragraphes intègrent espaces, virgules et points aux bons endroits.

Le code complet de cette démonstration tient sur moins de 200 lignes, incluant les données et des commentaires. En comparaison, le cas réel dont il est inspiré, pourtant moins complexe, en comporte sans les données plus de 600, la majorité gérant les détails que JSreal a automatisés.

5 Conception

La conception de JSreal a été influencée par :

- les objectifs du programme ;
- les particularités de JavaScript ;
- les particularités d'autres réalisateurs.

Cette section présente le fonctionnement interne de JSreal et les choix effectués lors de son développement.

JavaScript

Plusieurs réalisateurs utilisent comme spécification d'entrée une longue chaîne de caractères et peuvent donc développer leur propre "langage". Il aurait été intéressant, avec JSreal, de pouvoir entrer la phrase "la température est basse" ainsi :

```
[S [NP [D le] [N température]] [VP [V être] [A bas]]]
```

FIGURE 70: Spécification simple textuelle

Cette notation est utilisée dans plusieurs outils pour produire des arbres syntaxiques et aurait donc été familière auprès de plusieurs usagers. Il n'aurait que fallu la modifier quelque peu pour permettre d'assigner des traits et d'utiliser des variables. J'ai décidé d'utiliser des entrées semblables à cette notation, mais qui utilisent seulement du JavaScript plutôt que de requérir l'analyse de chaînes de caractères :

```
S(NP(D('le'), N('température')), VP(V('être'), A('bas')))
```

FIGURE 71: Spécification simple en JavaScript

Fichiers

Comme JSreal est écrit en JavaScript, il est très simple de le rendre disponible dans une page web : il suffit de charger deux fichiers, par exemple en ajoutant les lignes suivantes dans la partie `head` d'un document HTML.

```
<script src="http://daou.st/JSreal/lex-fr.js"></script>
<script src="http://daou.st/JSreal/JSreal.js"></script>
```

FIGURE 72: Chargement du lexique et du programme

Le fichier `lex-fr.js` est le lexique, qui contient des données en format JSON destinées à JSreal.

`JSreal.js`, quant à lui, contient le réalisateur, entièrement englobé dans une fonction auto-exécutante, une particularité de JavaScript. Cette structure assure que les nombreuses fonctions-outils que je déclare à l'intérieur de JSreal ne viennent pas polluer l'espace de noms de l'usager ; seules les fonctions que je choisis échappent au fichier.

Paramètres

JSreal commence par quelques paramètres que l'utilisateur est susceptible de vouloir modifier. Chaque paramètre est une propriété d'un objet nommé `param` modifiable à l'extérieur de JSreal.

Outils

Suivent ensuite quelques fonctions- et méthodes-outils, qui sont utilisées tout au long du code. Elles consistent surtout en des manipulations de chaînes, permettant par exemple de vérifier si une chaîne commence par une sous-chaîne particulière.

Quelques fonctions-outils retournent du code sous forme de chaîne de caractères ; à différent endroits de JSreal, j'utilise la fonction `eval`, native à JavaScript, pour exécuter le code décrit par la chaîne. Par exemple, la fonction `makeCont`, utilisée à une dizaine de reprises, intègre à une unité son contexte et crée automatiquement plusieurs variables qui correspondent à des traits demandés :

```
eval(makeCont('f','pe','n'))
```

FIGURE 73: Évaluation d'une chaîne retournée par une fonction

Sans cette fonction, je devrais manuellement écrire le code suivant :

```
cont = cont || {}
for (p in cont) if (cont[p]) this.cont[p] = cont[p]
var f = this.f()
var pe = this.pe()
var n = this.n()
```

FIGURE 74: Code écrit manuellement

5.1 Unités

En JavaScript, un objet n'est qu'un ensemble de propriétés et de méthodes. Dans JSreal, chaque unité, comme `A('bas')` ou `NP(D('le'), N('température'))`, a comme prototype le même objet : `JSrealE`, pour *JSreal Element*. Toutes les unités ont donc les mêmes propriétés possibles et peuvent utiliser les mêmes méthodes.

L'objet `JSrealE` n'est jamais utilisé directement par l'usager, qui manipule uniquement les unités qui en héritent.

Mots

Les unités de base sont les catégories de mots, `N`, `V`, `D`, `A`, `Pro`, `Adv`, `P` et `C`. Ces catégories sont énumérées dans une liste JavaScript pour qu'une boucle les intègre toutes comme fonctions globales dans le code, chacune retournant un objet `JSrealE` dont une propriété est la catégorie correspondante. Cette automatisation est possible car les fonctions globales ne sont que des propriétés de l'objet JavaScript tout en haut de la hiérachie, appelé `window` (la "fenêtre" d'un navigateur web), et JavaScript permet leur manipulation.

Pour créer un mot, il suffit d'utiliser la fonction correspondant à sa catégorie avec son lemme comme argument. Par exemple, pour créer l'adjectif "bas" et l'assigner à la variable `low` :

```
var low = A('bas')
```

FIGURE 75: Assignation d'adjectif à une variable

Les noms, adjectifs et déterminant peuvent aussi accepte un nombre comme argument au lieu d'un lemme.

Groupes

D'autres unités sont les groupes de mots, `NP`, `VP`, `AP`, `PP`, `CP`, `J`, `S` et `SP`. Ils ne sont pas déclarés en série comme les mots car chacun requiert une attention spéciale à sa création et ses méthodes.

Expressions temporelles

Les expressions temporelles se créent avec les fonctions `DT` et `DTR`.

`DT` est une unité de base dont l'argument peut être un objet `Date`, natif à JavaScript, ou une chaîne de caractères spécifique,[22] pour l'analyse de laquelle JSreal utilisera un objet `Date`. La date et l'heure d'un objet `DT` peuvent aussi être attribuées par des méthodes comme `y` (*year*) ou `h`.

`DTR` accepte deux arguments, chacun pouvant être un objet `DT` ou un argument que `DT` accepterait.

Lors de la réalisation, tant `DTR` que `DT` utilisent d'autres unités JSreal pour construire les expressions temporelles qui les représentent, s'assurant ainsi de respecter la grammaire et la syntaxe des groupes de mots dont elles font partie.

Balises HTML

Plusieurs balises HTML peuvent être crées dans JSreal avec des fonctions similaires à celles des mots et des groupes. Chacune de ces fonctions utilise la fonction be base DOM, qui retourne un objet `JSrealE` avec quelques propriétés et méthodes spécifiques aux balises. Comme pour les catégories, les fonctions associées aux balises sont déclarées en série à partir d'une liste.

Ajout de constituants

Toutes les manières d'ajouter un constituant à un groupe passent par la méthode `add`, lui donnant comme arguments les unités à ajouter. Cette méthode appartient à `JSrealE` et est donc exactement la même pour chaque groupe, qui accumule ses constituants pour le moment, ne les classant que lors de la réalisation.

Si la méthode `add` est appelée sur un mot comme `N` ou `A`, sa catégorie est utilisée avec `eval` pour créer une instance du groupe de mots qui l'utilise comme noyau (ici, respectivement `NP` ou `AP`) et y intégrer le mot et son ajout. Ainsi, les deux lignes suivantes sont exactement équivalentes :

```
A('bas').add(Adv('plus'))
```

FIGURE 76: Ajout d'adverbe à un adjectif ; changement en syntagme adjectival

```
AP().add(A('bas'), Adv('plus'))
```

FIGURE 77: Ajout d'adjectif et d'adverbe à un syntagme adjectival

Ajouter l'adjectif à l'adverbe aurait échoué, tentant de construire une syntagme adverbial avec la fonction `AdvP`, qui n'existe pas présentement ; même si elle existait, l'adjectif n'a aucune fonction possible dans un syntagme adverbial et la réalisation aurait donc été imprévisible.

En pratique, l'usager utilise les constituants d'un groupe comme arguments de la fonction associée à ce groupe ; JSreal appelle alors `add` sur ces arguments. Ainsi, la ligne de code suivante est exactement équivalente aux deux précédentes :

```
AP(A('bas'), Adv('plus'))
```

FIGURE 78: Adjectif et adverbe arguments de syntagme adjectival

5.2 Traits

Qu'elle soit un mot ou un groupe, une unité JSreal est un ensemble de propriétés JavaScript :

- la majorité des unités ont une catégorie ;
- un mot a un lemme ;
- un groupe a des constituants.

D'autres propriétés, appelées **traits**, peuvent être spécifiées par l'usager, le contexte ou le lexique.

Lexique

Un lexique a été essentiel dès les débuts de la conception de JSreal. J'ai choisi d'adapter celui de SimpleNLG-EnFr, en XML, qui comporte un peu moins de 4 000 entrées, dont la majorité des mots-outils et une sélection des mots les plus courants selon l'échelle orthographique Dubois-Buyse,[25] qui

étudie l'acquisition de vocabulaire par les enfants au fil de leur éducation. Cette source assure une bonne couverture de mots courants, mais elle n'est regrettablement pas adaptée aux besoins du web.

Un autre problème s'est révélé dès les débuts de la conception : le lexique de SimpleNLG-EnFr occupe 673 KB. Les entrées du mot "son", qui existe en tant que déterminant et nom, sont les suivantes :

```
<word>
  <base>son</base>
  <category>determiner</category>
  <feminine_singular>sa</feminine_singular>
  <plural>ses</plural>
  <liaison>son</liaison>
  <possessive/>
</word>
<word>
  <base>son</base>
  <category>noun</category>
  <gender>masculine</gender>
  <id>son_1</id>
  <plural>sons</plural>
</word>
```

FIGURE 79: Entrée XML de "son"

JSreal étant destiné à une utilisation web, la taille du lexique, pour un nombre donné d'entrées, doit être la plus faible possible. La seule option pratiquable a été de réduire à leur plus simple les expressions représentant la cinquantaine de traits recensés ; j'ai écrit un simple script Python pour parcourir le XML et le convertir en JSON selon mes besoins. Ma version du lexique utilise le lemme de chaque mot comme clé dans le dictionnaire (au sens informatique) ; plutôt que d'utiliser une balise `id` ou une propriété équivalente, je fais de chaque entrée une liste des mots possibles. Ainsi, l'entrée de "son" est contractée à cette simple ligne :

```
"son":[{c:"D",fs:"sa",p:"ses",l:"son",po:1},{c:"N",g:"m",p:"sons"}]
```

FIGURE 80: Entrée JSON de "son"

Cette concision réduit le lexique à seulement 197 KB (en comparaison, JSreal, sans minimisation, fait 67 KB) ; les identifiants minimalistes de traits se sont ensuite étendus au reste du programme. Étant en JSON, le lexique peut être utilisé directement par JavaScript, donc par JSreal, sans autre traitement ou analyse.

Méthodes

Les unités qui découlent de `JSrealE` gèrent leurs traits grâce à trois méthodes de base, où `p` et `val` sont des chaînes :

- `setP(p, val)` : assigne la valeur `val` au trait `p` de cette unité, puis retourne l'unité

- `getP(p)` : retourne la valeur du trait `p` de cette unité
- `delP(p)` : efface la valeur préalablement assignée au trait `p` de cette unité, si applicable, puis retourne l'unité

Les traits assignés à une unité par `setP` sont notés dans sa propriété `prop` :

```
var gf = N('gars').setP('g', 'f')
alert(gf.prop.g) // f
```

FIGURE 81: Assignation de propriété

Certains traits entraînent d'autres effets avec `setP` :

- les traits `sub`, `p` et `d` créent des nouveaux groupes de mots dont l'unité sera constituant ;
- le trait `pe` (personne) peut suffire à définir un pronom personnel.

La méthode `getP`, à la base, consulte plusieurs sources :

- si l'unité a le trait défini dans `prop`, sa valeur est retournée ;
- si l'unité a un contexte et que ce trait y est défini, sa valeur est retournée ;
- si l'unité est un mot et que ce trait est défini dans son entrée du lexique, sa valeur est retournée.

Mais `getP` est un des outils les plus détaillés et complexes de JSreal. Beaucoup de traits (dont la personne et le nombre) reçoivent un traitement particulier et la majorité des groupes redéfinissent cette méthode pour l'adapter à leurs particularités.

- `NP`, `AP`, `VP`, `S` et `SP` peuvent tirer des traits de certains de leurs constituants.
- `CP` et `J` utilisent des règles pour déterminer leur genre, nombre et personne à partir de leur coordonnant et de leurs autres constituants.
- `DOM` et les balises qui en découlent se réfèrent à leurs constituants pour la majorité des traits grammaticaux et syntaxiques.

Les méthodes `setP`, `getP` et `delP` sont utilisées à l'intérieur de JSreal, mais elles ne sont pas destinées à l'usager. Par exemple, la manière recommandée pour indiquer que "gars" est au féminin pluriel utilise directement les méthodes `g` et `n` :

```
N('gars').g('f').n('p')
```

FIGURE 82: Assignation de traits

La courte chaîne représentant un trait dans le lexique et à l'intérieur du programme est aussi une méthode de chaque unité. Tous les traits représentés dans le lexique, ainsi qu'une trentaine d'autres, font partie d'une liste ; une boucle JavaScript, en la traversant, fait de chaque trait une méthode de `JSrealE` qui, selon qu'elle a ou non un argument ou s'il est `null`, fait appel à `setP`, `getP` ou `delP`.

Cette manipulation concise des traits des unités, couplée à la simplicité de la création des unités, rend les spécifications de texte de JSreal extrêmement faciles à écrire et à lire.

5.3 Réalisation

Une fois la spécification créée à partir des unités et traits, il ne reste qu'à la réaliser. Elle peut varier en complexité, allant de la simple conversion en texte d'un nombre à la génération de la majorité du contenu d'une page web. Elle peut être découpée en petits morceaux intégrés un à la fois à la page ou être accumulée en une seule unité.

Arbre syntaxique

La réalisation d'une unité s'effectue en deux étapes, la première étant la constitution d'un arbre syntaxique par l'utilisation récursive de la méthode `tree` de `JSrealE`.

Chaque groupe de mots redéfinit la méthode `tree` pour lui-même. Quand `tree` est appelée sur un groupe donné, elle commence par attribuer à chacun de ses constituants une fonction selon sa catégorie et, parfois, d'autres facteurs. Par exemple, dans un syntagme verbal (`VP`) le premier verbe rencontré comme constituant sert de noyau et tout verbe éventuel en devient un complément.

L'ordre dans lequel les constituants d'un groupe lui ont été ajoutés affecte leur ordre dans le groupe seulement s'ils occupent la même fonction. Le trait `fct` permet à l'usager de spécifier manuellement la fonction d'un constituant, par exemple pour faire de "demain" le sujet d'une phrase ; le même effet peut être obtenu par des méthodes `add` modifiées, comme `addHead`.

Ensuite, le groupe intègre les traits reçus d'un éventuel contexte et amasse ceux qu'il devra transmettre à ses constituants.

Finalement, le groupe utilise des méthodes spécialisées de `JSrealE` pour construire son arbre. Il :

- copie chaque constituant pour éviter de modifier l'original ;
- ordonne chacun selon sa fonction dans le groupe ;
- appelle `tree` sur chacun, leur passant les traits appropriés à leur fonction.

Contrairement à plusieurs autres réalisateurs, JSreal n'utilise pas de "patrons" explicites pour ses groupes : chaque groupe se contente d'ordonner ses constituants selon des règles codées manuellement en JavaScript. La plupart de ces règles sont simples, mais certaines sont plus détaillés, par exemple celles de `VP` pour les pronoms personnels :

- si un constituant est un pronom personnel, son lemme change pour la forme appropriée comme complément d'objet direct et il est placé devant le verbe ;
- si un constituant est un pronom personnel introduit par la préposition "à", il est remplacé par le pronom personnel de même personne et nombre et approprié comme complément d'objet indirect et il est placé devant le verbe, mais après celui du point précédent ;
- si le verbe est à l'impératif, l'emplacement des pronoms personnels change et ils sont liés au verbe par des tirets.

Pour les mots, feuilles de l'arbre syntaxique, la méthode `tree` a généralement comme seul effet de leur transmettre le contexte du groupe dont ils font partie, si applicable, et met donc fin à la récursion.

Réalisation finale

Une fois l'arbre syntaxique constitué, JSreal peut réaliser l'unité.

Un groupe de mots appelle récursivement `real` sur chacun de ses constituants, dans l'ordre, et sépare les résultats par des espaces, dans une chaîne de caractères. Certains traits et groupes peuvent éliminer des espaces ou ajouter de la ponctuation, voire sauter une unité complète.

À l'atteinte d'un mot, celui-ci reçoit une attention particulière pour déterminer sa forme finale.

- Les nombres peuvent être mis en toutes lettres, puis déclinés quand approprié.
- Si un mot est invariable ou n'a aucun genre, nombre ou autre trait susceptible d'altérer sa forme, il retourne son lemme.
- Dans tous les autres cas, le lexique est consulté au cas où il indiquerait une forme irrégulière.
- Les noms, adjectifs, déterminants et pronoms se déclinent selon les règles appropriées à leur catégorie et terminaison.
- Les verbes se conjuguent selon les règles.
- Les mots suceptibles d'élision, de liaison ou de contraction traversent l'arbre pour vérifier ce qui les suit.

Une fois qu'une unité a produit une chaîne de caractères, JSreal vérifie de quoi elle est entourée, incluant la ponctuation et les balises HTML et leurs attributs, et si elle est suivie d'un espace. Les éléments appropriés sont concaténés à la chaîne dans un ordre particulier et elle est retournée comme réalisation.

Appel

À la base, une unité n'est qu'un objet `JSrealE` ; c'est la méthode `real` qui en demande la réalisation, menant à la contruction de l'arbre syntaxique et à la linéarisation.

Quand un objet JavaScript est utilisé comme s'il était une chaîne de caractères, JavaScript utilise la méthode `toString` de cet objet, si elle existe ; par convention, cette méthode retourne une chaîne représentative de l'objet. La méthode `toString` de l'objet `JSrealE` retourne sa propre réalisation, passant par `real`.

La méthode `node` convertit une unité en élément DOM. Si l'unité découle de la fonction `DOM` de JSreal, l'élément correspondant est retourné ; sinon, c'est un élément DOM textuel qui est retourné.

Les méthodes `toID` et `addToID` envoient ou ajoutent la réalisation de l'unité à l'élément DOM qui a l'identifiant spécifié en argument.

5.4 Conclusion

La conception de JSreal répond bien aux objectifs du projet.

Intégrer JSreal à une page web est aussi simple que pour n'importe quelle autre bibliothèque : il suffit d'insérer un lien vers son lexique et lui-même. Ce lexique, devant être de taille aussi réduite que possible pour être chargé rapidement sur le web, indique les traits des mots en utilisant le moins de caractères possible.

Les spécifications de JSreal sont plus accessibles que celles des autres réalisateurs, leur entrée étant concise et flexible, gardant l'accent sur les unités, les traits et les lemmes. Des méthodes simplifient la gestion des déterminants, des prépositions et des subordonnées.

Cette accessibilité est permise par l'utilisation d'un seul objet, `JSrealE`, comme base de toutes les unités. Toutes les fonctionnalités de JSreal passent par cet objet, qui partage ses nombreuses méthodes aux fonctions qui en découlent, comme `Pro` et `NP` ; ainsi, toutes les unités ont, par exemple, un genre et une réalisation, accédés de la même manière.

La particularité qu'a JavaScript de permettre un accès sans limite à tous les objets permet de déclarer en série les fonctions globales correspondant à plusieurs unités et les méthodes correspondant à tous les traits. La capacité de JavaScript à interpréter une chaîne de caractères comme du code est utilisée pour éviter de devoir entrer du code répétitif manuellement.

L'organisation par groupes de mots permet à JSreal de propager aisément les traits entre ses unités. Sa construction d'un arbre syntaxique pour ordonner les unités permet qu'il soit traversé pour les besoins de l'élision, la liaison et la contraction et rend sa linéarisation aussi simple que quelques appels récursifs qui vérifient en même temps de quels caractères chaque unité est entourée.

L'usager dispose finalement d'une variété de manières d'utiliser les structures JSreal qu'il construit, que ce soit en les insérant comme chaînes de caractères ou en les convertissant en éléments DOM.

6 Conclusion

JSreal a été créé dans le but d'être intégré dans la programmation web, tant comme outil à utiliser pour obtenir plus facilement certaines expressions que comme manière de générer des phrases et documents complexes. Les exemples donnés au cours de ce mémoire démontrent l'atteinte de cet objectif : JSreal est simple à intégrer, à apprendre et à utiliser, allège significativement la tâche du programmeur et peut produire une variété appréciable de contenu.

En comparaison aux autres réalisateurs de texte, JSreal intègre peu de théorie syntaxique, mais cette faiblesse apparente facilite son utilisation, l'usager n'ayant qu'à correctement regrouper ses mots, par exemple en syntagmes nominaux et en phrases. JSreal étant le seul réalisateur programmé dans un langage web, il n'a pas vraiment d'alternative dans ce milieu ; de plus, il s'inscrit dans le paradigme d'accessibilité des bibliothèques JavaScript.

En tant que preuve de concept, JSreal démontre que le web peut définitivement bénéficier d'un réalisateur qui lui est spécifique. Les outils tels que la gestion des nombres, des expressions temporelles et des balises HTML, quoique disponibles ailleurs, gagnent à être intégrés à une bibliothèque qui leur permet d'interagir ensemble et avec du contenu en langage naturel et vice-versa.

Travaux futurs

Malgré les résultats déjà obtenus par JSreal, il comporte quelques lacunes qui compliquent son utilisation.

En particulier, JSreal permet seulement les propositions déclaratives positives. En l'absence de soutien pour les adverbes interrogatifs et l'inversion du sujet, les seules phrases interrogatives possibles s'obtiennent en substituant le point à la fin pour un point d'interrogation. Et aucune structure JSreal raisonnable ne permet de réaliser correctement une proposition négative.

Le lexique actuel, quoique fonctionnel, n'a pas été développé spécifiquement pour JSreal, alors il inclut des traits non utilisés et des mots sans traits hormis la catégorie et comporte plusieurs redondances dans ses formes irrégulières, qui ne sont pas toujours vraiment irrégulières : par exemple, le pluriel de "appétit" y est inutilement indiqué comme "appétits". Nettoyer le lexique ou le reconstituer intégralement permettrait de réduire encore plus sa taille pour une quantité d'informations donnée. Aussi, comme le lexique est beaucoup plus volumineux que le reste du programme, il pourrait être épuré pour chaque projet, omettant par exemple le vocabulaire et les temps de verbes peu susceptibles de servir.

Les autres lacunes de JSreal sont moins graves, concernant notamment la ponctuation dans les coordinations, les traits plus sémantiques des verbes et les locutions déterminatives et adverbiales.

Une autre considération quand au futur de JSreal serait une version anglaise, qui rejoindrait un nombre accru d'utilisateurs.

Au-delà du perfectionnement du programme lui-même, des travaux futurs pourraient porter sur la profondeur viable d'un réalisateur complet pour le web, en particulier quant à sa complexité d'utilisation et à sa vitesse de chargement et de fonctionnement. Quelle combinaison de simplicité, légèreté et rapidité feraient de ce réalisateur un outil incontournable dans les projets de génération de texte sur le web ?

Références

[1] Daoust, Nicolas. “JSreal”. JSreal.
<http ://daou.st/JSreal> 10 sep 2013

[2] Daoust, Nicolas. “Documentation”. JSreal.
<http ://daou.st/JSreal/#/Documentation> 10 sep 2013

[3] Daoust, Nicolas. “Test – JSreal”. JSreal.
<http ://daou.st/JSreal/Test> 10 sep 2013

[4] Daoust, Nicolas. “Tests”. JSreal.
<http ://daou.st/JSreal/#/Annexes/Tests> 10 sep 2013

[5] Daoust, Nicolas. “Traits”. JSreal.
<http ://daou.st/JSreal/#/Annexes/Traits> 10 sep 2013

[6] Reiter, Ehud ; Dale, Robert. Building Natural Language Generation Systems. Cambridge University Press, 2000.

[7] McDonald, David D. “Natural Language Generation” dans Encyclopedia of Artificial Intelligence, John Wiley and Sons, New York, 2e édition, pp. 983–997

[8] “Table of NLG systems”. NLG Systems Wiki.
<http ://www.nlg-wiki.org/systems/Table_of_NLG_systems> 9 sep 2013

[9] Bourbeau, L. ; Carcagno, D. ; Goldberg, E. ; Kittredge, R. ; Polguère, A. “Bilingual generation of weather forecasts in an operations environment” dans *Proceedings of the 13th International Conference on Computational Linguistics*, pp. 318–320, 1990.

[10] Lavoie, Benoit ; Rambow, Owen. ; Reiter, Ehud. “Customizable Descriptions of Object-Oriented Models” dans *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pp. 253–256, 1997.

[11] Milosavljevic, Maria ; Dale, Robert. “Text Generation and User Modelling on the Web”. Macquarie University.
<http ://web.science.mq.edu.au/~mariam/papers/um96/nlg-um-www.html> 9 sep 2013

[12] Vaudry, Pierre-Luc. “SimpleNLG-EnFr”. Département d’informatique et de recherche opérationnelle de l’Université de Montréal.
<http ://www-etud.iro.umontreal.ca/~vaudrypl/snlgbil/snlgEnFr_english.html> 10 sep 2013

[13] Vaudry, Pierre-Luc ; Lapalme, Guy. “Adapting SimpleNLG for bilingual English–French realisation” dans *Proceedings of the 14th European Workshop on Natural Language Generation*, pp. 183–187, 2013.

[14] “KPML one-point access page”. Universitat Bremen.
<http ://www.fb10.uni-bremen.de/anglistik/langpro/kpml/README.html> 9 sep 2013

[15] “RealPro”. CoGenTex, Inc.
<http ://www.cogentex.com/technology/realpro/index.shtml> 10 sep 2013

[16] Lavoie, Benoit ; Rambow, Owen. “A Fast and Portable Realizer for Text Generation Systems” dans *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pp. 265–268, 1997.

[17] Elhadad, Michael. “SURGE : A Syntactic Realization Grammar for Text Generation”. Computer Science — Ben Gurion University of the Negev.
<http ://www.cs.bgu.ac.il/surge> 9 sep 2013

[18] “SimpleNLG”.
<http ://code.google.com/p/simplenlg> 9 sep 2013

[19] Gatt, Albert ; Reiter, Ehud. “SimpleNLG : A realisation engine for practical applications” dans *Proceedings of the 12th European Workshop on Natural Language Generation*, pp. 90–93, 2009.

[20] “Downloadable NLG systems”. Association for Computational Linguistics Wiki.
<http ://aclweb.org/aclwiki/index.php ?title=Downloadable_NLG_systems> 9 sep 2013

[21] Grevisse, Maurice. Le Bon Usage : Grammaire française. Éditions Duculot, Paris-Gembloux, 11^{e}édition, 1980.

[22] “Date.parse”. Mozilla Developer Network.
<https ://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/parse> 9 sep 2013

[23] Daoust, Nicolas. “Événements”. Luc Lightbringer.
<http ://www.luclightbringer.com/index.php ?ong=4> 9 sep 2013

[24] Daoust, Nicolas. “Démonstration – JSreal”. JSreal.
<http ://daou.st/JSreal/Demo> 10 sep 2013

[25] “Echelle orthographique Dubois-Buyse”. Ressources francophones de l’Education.
<http ://o.bacquet.free.fr/db2.htm> 9 sep 2013

[26] Daoust, Nicolas. [Lexique JSreal].
<http ://daou.st/JSreal/lex-fr.js> 10 sep 2013

[27] Daoust, Nicolas. [Code JSreal].
<http ://daou.st/JSreal/JSreal.js> 10 sep 2013