Chapter 1
JSREAL: A text realizer for web programming

Nicolas Daoust and Guy Lapalme

Abstract The web is constantly growing and its documents, getting progressively more dynamic, are well-suited to presentation automation by a text realizer. Current browser-based information display systems have mostly focused on the display and layout of textual data, restricting the generation of nonnumerical informations to canned text or formatted strings. We describe JSREAL, a French text realizer implemented in Javascript. It allows its user to build a variety of French expressions and sentences, combined with HTML tags to easily integrate them into web pages to produce dynamic output depending on the content of the page.

1.1 Context

Natural language generation (NLG) is often used to textually describe massive datasets, making the most of computers' ability for rapid analysis. Even in the cases where experts could interpret the data by themselves, it can be useful to present summaries of other pertinent aspects of the data to non-experts.

Beyond rapid analysis, another advantage of computers over humans is their tireless efficiency at carrying out routine tasks. Many people, in the course of everyday work, have to repeatedly express information of which only certain details change from one instance to another, for example in customer support. NLG can automate a significant part of that type of textual production, only requiring a human to supply some important aspects and thus saving considerable time and producing consistent grammatically correct output.

Following the classical architecture of Reiter and Dale [9], NLG involves three stages: macro-planning (content choice and organization), micro-planning (abstract specification of linguistic form by means of finding appropriate referring expressions, doing some aggregation and choosing the words and grammatical forms) and

Nicolas Daoust e-mail: n@daou.st · Guy Lapalme e-mail: lapalme@iro.umontreal.ca
RALI-DIRO, Université de Montréal, C.P. 6128, Succ Centre-Ville, Montréal, Canada, H3C 3J7

realization (conversion into concrete spoken or written form). In this paper, we focus on this last step, which is further subdivided into surface realization (the sequence of words and sentences) and physical presentation (HTML presentation).

1.2 Realization

All programming languages can produce textual output with so-called *print* statements and most of them allow some flexibility with *formatting* schemes, some of them being quite sophisticated, but no programming language provides a complete text realizer. So when more fluent text must be generated (especially in morphologically rich languages such as German or French), this requires a dedicated specialized realizer. Such realizers feature patterns for phrases, sentences and documents, but only a few of their aspects (e.g. numeric values or number of different values) are modifiable and they would thus require a thorough overhaul to be adapted for another project. Whereas a complete realizer encodes a large diversity of words, phrases and grammatical rules, a specific realizer only covers whatever few words and structures it needs to, processing them quite naïvely.

Some realizers, such as KPML [15], SURGE [5] and REALPRO [7, 16], are based on complex syntactic theories, taking into account many details in the construction of sentences, which allows powerful realizations. However, that complexity hinders their ease of use: writing specifications for them requires an intimate knowledge of the underlying theory, which is not the case of most programmers.

In fact, most existing realizers are considered so convoluted that SIMPLENLG [6, 17], as its name implies, defines itself by its ease of learning and of use [13]. Words, phrases and other structures being Java objects, they can be created and manipulated intuitively by a programmer and they are easily integrated into a Java project. While its fundamentals do not allow for realizations as powerful as some other realizers, they largely suffice for most use. SIMPLENLG has been used in a variety of text generation projects such as described by Portet *et al.* [8].

Some of the existing realizers, like KPML and REALPRO, are technically capable of producing web output, such as HTML; KPML even allows the addition of rudimentary HTML tags to words or phrases. However, no realizer is written in a web programming language, such as JavaScript or PHP, and integrating another language on a web page, while feasible, is only practical in select cases. In any case, this means that web-based text generation projects are currently better-served by their own custom realizers.

1.3 JSREAL

JSREAL (JavaScript REALizer) is a French text realizer that generates well-formed expressions and sentences and can format them in HTML to be displayed in a browser. It can be used standalone for linguistic demonstrations or be integrated into complex text generation projects. But first and foremost, JSREAL is aimed at web developers, from taking care of morphology, subject-verb agreement and conjugation to creating entire HTML documents.

As its name indicates, JSREAL is written in JavaScript, a programming language that, when used in a web page, runs in the client browser. A web programmer that wishes to use JSREAL to produce flexible French textual output only needs to add two lines in the header of the page (one for loading the French lexicon and one for program), similarly as what is done for other browser frameworks such as JQUERY.

The specifications of JSREAL are similar to those of SIMPLENLG: they are programming language instructions that create data structures corresponding to the constituents of the sentence to be produced. Once the data structure (a tree) is built in memory, it is traversed (a phase called *realization)* to produce the list of tokens that will form the sentence. This data structure is built by function calls whose names are the same as the symbols that are usually used for classical syntax trees: for example, `N` to create a *noun* structure, `NP` for a *Noun Phrase*, `V` for a *Verb*, and so on.

So instead of creating specifically formatted strings, the programmer uses usual JavaScript instructions that can be freely manipulated and that are particularly concise, resembling the syntactic trees of the phrases they represent (see Figure 1.1).

```
S(NP(D('le'),
     N('chat')).n('p'),
  VP(V('manger'),
     NP(D('un'),
        N('souris'))))
```

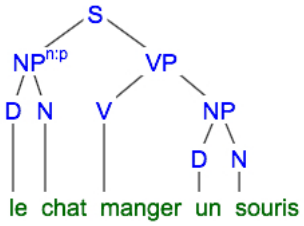

Fig. 1.1 A sample JSREAL expression and the corresponding tree for the sentence Les chats mangent une souris (*The cats eat a mouse*).

Words in upper-case are function calls that create a data structure from the values returned by their parameters. As all these values are objects whose features can be modified by function calls that are specified using the *dot-notation* such as `.n('p')` in the second line of the left of Figure 1.1. In this case, this means that the number of the noun phrase should be plural.

JSREAL's capabilities are similar to other realizers':

- its lexicon defines word categories, genders, numbers, irregularities and other features;

- its knowledge of French morphologic rules allows it to use the appropriate word forms, such as plurals and conjugations;
- its knowledge of French syntactic rules allows it to properly order words in a sentence, transmit agreement (this is why the verb manger (*eat*) is plural in Figure 1.1) and carry out other interactions.

JSREAL seamlessly integrates other useful tools, such as the spelling out of numbers and the wordings of temporal expressions. Since it produces web content, it uses HTML formatting for paragraphs, lists and similar structures and can manipulate them in detail and make sure that they do not interfere with the proper processing of words within a sentence.

Specifications can be as complex as needed, as shown in figure 1.3, where a title including an inline image is followed by a paragraph formed by a sentence coordinating three phrases (each of them being full sentences). Links, commas and the conjunction et (*and*) are appropriately inserted.

```
Div(H2(N('propos de ').p('à'),
     Img().class('inline').src('MZ.png')),
    P(CP(S(N('nom').d('p3'),
           VP(V('être'),
              N('Michael Zock').tag('b'))),
         S(Pro('il'),
           VP(V('être'),
              NP(N('pionnier').d('i'),
                 PP(P('de'),
                    N('génération').d('d').add(
                      A('automatique').pos('post'),
                      N('texte').p('de'))
                        .href('http://www.nlg-wiki.org/systems/'))))))
            .a2(''),
         Br(),
         S(Pro('vous'),
           VP(V('pouvoir'),
              VP(V('contacter'),
                 Pro('lui'),
                 Email('michael.zock@lif.univ-mrs.fr')
                          .p('à')))))))
```

À propos de 

Son nom est **Michael Zock**, il est un pionnier de la génération automatique de texte et vous pouvez le contacter à michael.zock@lif.univ-mrs.fr.

Fig. 1.2 A more complex specification to produce HTML output

1.4 The design of JSREAL

The design of JSREAL was influenced by the style of programming usually used in JavaScript and web development: that means that the specifications are relatively terse and rely on run-time checking rather than a compile-time type checking of the program used in SIMPLENLG .

Previous realizers such as KPML and RealPro take as input files written in a special formalism. Although these realizers can be run from other programs, they mostly behave as black boxes. JSREAL's specifications are inspired by a very simple syntactic tree notation already used in some phrase structure grammar formalisms as shown in figure 1.1; however, to avoid the black box problem, JSREAL specifications are made up of JavaScript object-creating functions, similar to SIMPLENLG Java objects.

JavaScript objects are simply a set of properties and methods. In JSREAL, words, phrases and formatting elements are called `units`, and they all have the same object as prototype, thus sharing methods and possible properties.

The basic units are the *word* categories: noun (`N`), verb (`V`), determiner (`D`), adjective (`A`), pronoun (`Pro`), adverb (`Adv`), preposition (`P` which depending on the context is also used for paragraphs) and conjunction (`C`). Each of the corresponding functions (such as `N` and `V`) takes a single lemma as only argument and return a JavaScript object with the appropriate properties and methods. These words can be used standalone or in phrases.

Phrases are higher-level units and are created with functions such as `NP` (noun phrase) and `VP` (verb phrase). They take other units as arguments, incorporating them as constituents, but can also receive constituents by the `add` method. An empty phrase (one that has no constituent) is skipped during realization. *Sentences* and clauses are created with `S`, and if not subordinated, are automatically added a capital and punctuation.

JSREAL's French lexicon is a direct adaptation of that of SIMPLENLG-ENFR [11, 10], a bilingual French and English version of SIMPLENLG. That lexicon is in XML and consists of slightly less than 4,000 entries, including most function words and a selection of the most common content words according to the Dubois-Buyse orthographic scale, which studies children's learning as they progress through school. Tools allow the user to add words to the lexicon and edit existing entries.

SIMPLENLG-ENFR's lexicon occupies 673 KB, which is relatively large by web standards. For a given number of entries, JSREAL's lexicon must be as light as possible, so we wrote a Python script to produce a JSON version of the dictionary. This amount to creating an object where each lemma is a key. Figure 1.4 shows the resulting entry for son (*his* as adjective, *sound* as a noun). This reduced the lexicon to 197 KB. Since JSON is native to JavaScript, it is used directly and combined with the dictionary structure, access speed is highly optimized.

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

Fig. 1.3 JSON entry for the French word `son` with two uses: a determiner (*his* or *her)*, the singular feminine and plural forms are given; a noun (*sound*), its gender and plural forms are given.

1.4.1 Features

Every unit has multiple properties, such as a category, a lemma (in the case of a word) or constituents (in the case of a phrase). Other properties, called *features*, can be found in the lexicon or specified by the user or the context. There are about 70 features in total and each is represented in JSREAL by a concise set of characters as shown in Table 1.1:

Feature	Meaning	Some allowed values
`g`	gender	`m`, `f`
`n`	number	`s`, `p`
`f`	verb tense, aspect and mood	`p`, `i`, `pc`
`pe`	person (4 to 6 are converted to 1 to 3, but plural)	`1`, `5`
`pos`	position in a phrase	`beg`, `pre`, `mid`
`fct`	function in a phrase	`head`, `comp`, `sub`
`d`	determiner (and makes the unit a noun phrase)	`d`, `i`, `dem`
`sub`	complementize (and makes the clause subordinate)	a word such as `que`
`cap`	capitalization	`1`
`a`	what comes after (generally punctuation)	`.`
`co`	comparative form	a word such as `mieux`
`ns`	no space after	`1`
`ell`	ellipsis of this unit	`1`

Table 1.1 Sample values allowed for the main features. All features and their possible values are detailed in the online documentation of JSREAL [2].

The user can access features by calling the relevant method of a unit, which is the same string that identifies that feature everywhere else, as shown in earlier examples such as Figure 1.1. Usually, the user will input the desired value as an argument or `null` to remove a previously set feature; in those cases, the method also returns the unit it is called on, allowing a sequence of features to be set. Omitting the argument makes the method return the current value of the feature.

One of the most complex parts of JSREAL is its ability to compute the value of a feature. In general, JSREAL first checks if the user has defined the feature, then if the context imposes a value, then if a value is found in the lexicon, and finally outputs `undefined`, letting the default processing be applied.

Many features follow detailed grammatical rules, in particular number and person, as shown in the two examples of figures 1.4. Some phrases redefine the retrieval of some features. Noun, adjective and verb phrases, as well as clauses and coordinations, can extract features from their constituents. Formatting tags, to be discussed

```
S(CP(C('ou'),
     Pro('moi'),
     Pro('toi'),
     Pro('elle')),
  V('manger'))

S(CP(C('et'),
     Pro('moi'),
     Pro('toi'),
     Pro('elle')),
  V('manger'))
```

Fig. 1.4 The first expression generates Moi, toi ou elle mange. (*I, you or she eats.*) in which the verb manger (*eat*) stays singular because of the disjunction ou (*or*). The second example generates Moi, toi et elle mangeons. (*I, you or she eat.*) with the plural verb because of the conjunction et (*and*). In both cases, the verb is conjugated at the first person because French grammar rules that the verb agrees with the *lowest* person.

in section 1.4.2, always defer to their constituents for grammatical features, having none themselves.

Some JSREAL units can accept a JavaScript number instead of a lemma. A number can be used as a noun with no special results; as a determiner, it correctly propagates its number to its associated noun; as an adjective, it becomes an ordinal if associated to a name. With the `num` feature or by changing a general setting, the user can ask JSREAL to spell out a number or format it in another way. The first four lines of Table 1.2 show a few examples.

JSREAL expression	**text output**
`S(V('appeler').f('ip').pe(5)` `.add(N(18005556426).num('t')))`	Appelez 1-800-555-6426. (*Call 1-800-555-6426.*)
`N(123456).num("l")`	cent vingt-trois mille quatre cent cinquante-six (*one hundred twenty-three thousand four hundred fifty-six*)
`N('enfant').d(3).num("l")`	trois enfants (*three children*)
`S(A(1).num('o').d('d'),` `V('gagner'))`	Le premier gagne. (*The first wins.*)
`DT().y(2014).m(7).d(14)` `.h(16).min(29)`	le lundi 14 juillet 2014 à 16 h 29 (*Monday July 14th at 16:29*)

`S(Pro('je'),`
`  VP(V('être').f("f"),`
`  Adv('là')),`
`  DTR(DT().y(2014).m(10).d(14),`
`  DT().y(2014).m(10).d(17)`
`  ).noDay(true).pos('end')`
`)`
	Je serai là du 14 au 17 octobre 2014.
	(*I will be there from October 2014 14 to 17.*)

Table 1.2 Examples of number and date JSREAL expressions and the corresponding output (and English translation)

With the DT function, JSREAL can automate the writing of dates and times, considering days of the week and redundancies and using expressions such as *yesterday* and *noon*. The date and time can be specified either by a JavaScript Date object (passed as argument), by specific date strings or by methods such as y, d and min. The DTR function can make a range from two temporal expressions, removing redundancies. Both DT and DTR can be used as constituents in other units. The last two lines of Table 1.2 shows the creation of temporal expressions with their output.

1.4.2 Formatting

Being web-based, JSREAL uses HTML for its formatting; the most basic is the use of the P function to join sentences in a paragraph. Many other HTML tags, like H4, A and B, also have their own functions, and DOM allows the creation of custom tags. Tags can also be added to units with the tag method. Figure 1.3 shows a paragraph preceded by a title and image. In most cases, tags are transparent to grammatical features, so a word can be bolded in the middle of a sentence with no adverse effect.

Another important part of HTML formatting are tag *attributes*. As with grammatical features, many attributes, such as class, href and src, have their own methods. Custom attributes can be assigned with the attr method.

Some formatting elements do further processing:

- H1 to H6 properly capitalize their content;
- UL and OL add li tags to their constituents;
- adding a href attribute also adds the a tag to the unit;
- HL and Email automate link input, adding the appropriate href attribute.

1.4.3 Realization

The final realization of a unit can be obtained in several ways:

- using a unit where JavaScript expects a string realizes that unit as a string;
- the node method makes an HTML element from the realization;
- the toID and addToID place the realization in the web page, at the selected ID.

Once the user asks for the realization of a unit, a syntactic tree is built recursively, down to individual words (and independent HTML tags). In each phrase, constituents are ordered according to their function and features and then by their order of input. Phrases then compute their relevant features (as discussed in section 1.4.1) and pass them on to their constituents according to their function; for example, a noun phrase can get its number from its determiner and transmit it to its noun, like in the trois enfants (*three children*) example of Table 1.2.

With the syntactic tree complete and all features determined, units are realized from the bottom up. Words return their final form, which can depend on their features and, in cases such as contracting at the request of the neighbouring words (e.g. le homme (*the man*) will be transformed to l'homme). Phrases then join their constituents with spaces and surround them with punctuation and HTML tags.

1.5 Use case

As a use case of JSREAL, we want to maintain a web page listing upcoming events that Alice, Robert and I offer the public. Since the information always uses similar syntactical structures, we want to generate it automatically.

We have at our disposal a dataset, shown in figure 1.5, that collects relevant information about the events, namely their date and time, place, category, participants and who to contact for reservations. The data can be assumed to be made available to the web application, it could be the result of parsing a calendar or querying a database. A sample of the desired output is shown in figure 1.5.

```
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'}
]
```

Fig. 1.5 Event list input in JSON format. Each event is a Javascript object with the following fields: `date`: the date of the event; `ville`: the town in which the event will be held; `cat`: the category of the event (`at`: atelier (*workshop*); `cs`: consultation, `cf`: conference); `h`:time of the event; `attr`: attribute of the event, `tit`:title of the event; `part`: initial of the participant; `res`: initial of the contact person for reservation

Since our data is already collected, there is no content determination step in this text generation project; we can thus start directly with the microplanning phase.

1. We notice that `Alice` and `Robert` are not in the lexicon, so we add them, specifying their gender.
2. We prepare additional information, such as the contact details for each participant.
3. We go through the data event by event, accumulating for each group the title, date, participants, place, events and reservation details.
4. We integrate the result into the web page (see Figure 1.5).

Atelier à Laval

Le mercredi 25 septembre 2013 à 9 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 d'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, à l'hôtel Castel, au 901 rue Principale, pour des consultations privées et deux ateliers.

- 30 septembre à 9 h: atelier classique « Principes de réalisation » avec Alice
- 2 octobre à 9 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 15 h: nouvelle conférence « Pourquoi la réalisation? »
- 3 octobre à 9 h: atelier « Planification et réalisation »

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

Fig. 1.6 Output of a list of events. For the full code, see [3]

JSREAL simplified many aspects of text generation in this project. It:

- added capitals, spaces and punctuation where appropriate;
- took care of verb agreement according to the number and person of the participants;
- created and used clauses, coordinations and lists regardless of if they had any constituents, skipping them seamlessly if they were empty;
- expressed dates and times naturally, looking up the days of the week by itself;
- took care of all the HTML formatting.

The example in this demonstration is fictitious, but it is inspired by a real text generation project [1]. The original project, yet less complex, numbered more than 600 lines of Javascript code, whereas the program written for this demonstration has under 200 lines (not counting the 2000 lines of JSREAL of course).

Starting from concise data instead of manually typing the results has other advantages: it would be easy to use the data to make a summarized table of events in addition to the verbose list (in fact, the original project had such a table). Also, in a future version of JSREAL, bilingual realization could be possible, allowing us to present events to both French and English clients with little more hassle.

1.6 Other examples

We have also developed other web based applications whose output is briefly illustrated here. These applications can all be tried online on the RALI website[1]:

- The main advantage of using a text realizer is the fact that the same pattern of code can be reused provided it is appropriately parameterized. Figure 1.6 shows how a conjugation table for a French verb can be created with the corresponding display in the browser.
- A demo page (Figure 1.6) that can be used to build a sentence with a simple syntactic structure and modify some of its components to see how it is realized in French and the corresponding JSREAL expression. It was useful to quickly test some features during its development, but it could also be helpful to learn how to write simple French sentences or to learn how to JSREAL code.
- We have also developed a JSREAL development environment (Figure 1.9) integrating the ACE javascript editor[2] in which a JSREAL expression can be entered on the left and that shows the corresponding syntactic tree on the right. The realization of the expression is shown at the top of the tree, but it is also possible to get intermediate realization by clicking on any node of the tree.
- Dynamic realization is illustrated with a variant on the `Exercices de style` of Raymond Queneau which are a classical word play in French. The original book (published initialy in 1947) showed 99 stylistic variations on a text describing a situation in which the narrator gets on the *S* bus, witnesses a dispute between a man and another passenger, and then sees the same person two hours later at the Gare St-Lazare getting advice on adding a button to his overcoat. It has since then been adapted in various languages and in multiple styles. The text at the top of Figure 1.10 was produced with JSREAL, but by selecting from the menu, we can get a variant in which all the nouns representing persons have been made plural. The nouns that have been changed are underlined in the resulting text shown at the bottom of Figure 1.10; the verbs that have these nouns as subjects have also been changed, but are not underlined.

[1] http://rali.iro.umontreal.ca/rali/?q=fr/jsreal-realisateur-de-texte

[2] http://ace.c9.io

```
function addTable(verbe,temps){
    // temps is an array of two element arrays each giving
    // the name of a tense and the JSreal code for feature f
    var tableau=$("tableau");    // find the table element
    var row=TR();                // create a new row
    for (var t=0;t<temps.length;t++) // fill the title of the table
        row.add(TH(N(temps[t][0])))
    tableau.appendChild(row.node()); // add it to the table
    // generate a row for the 6 persons (3 singular and 3 plural)
    for(var p=1;p<=6;p++){
        row=TR();
        for(var t=0;t<temps.length;t++){ // a row at 3 tenses
            var v=S(Pro().pe(p),V(verbe).pe(p).f(temps[t][1]));
            row.add(TD(v.a("")));
        }
        tableau.appendChild(row.node());
    }
}

function conjuguer(verbe){
  H2(V(verbe)).toID("verbe");
  addTable(verbe,[["Présent","p"],
                  ["Imparfait","i"],
                  ["Futur simple","f"]]);
}
```

Tableau de conjugaison

Conjuguer [échanger]

Échanger

Présent	Imparfait	Futur simple
J'échange	J'échangeais	J'échangerai
Tu échanges	Tu échangeais	Tu échangeras
Il échange	Il échangeait	Il échangera
Nous échangeons	Nous échangions	Nous échangerons
Vous échangez	Vous échangiez	Vous échangerez
Ils échangent	Ils échangeaient	Ils échangeront

Fig. 1.7 JSREAL code for producing a conjugation table and its display when called on the verb échanger (*exchange*).

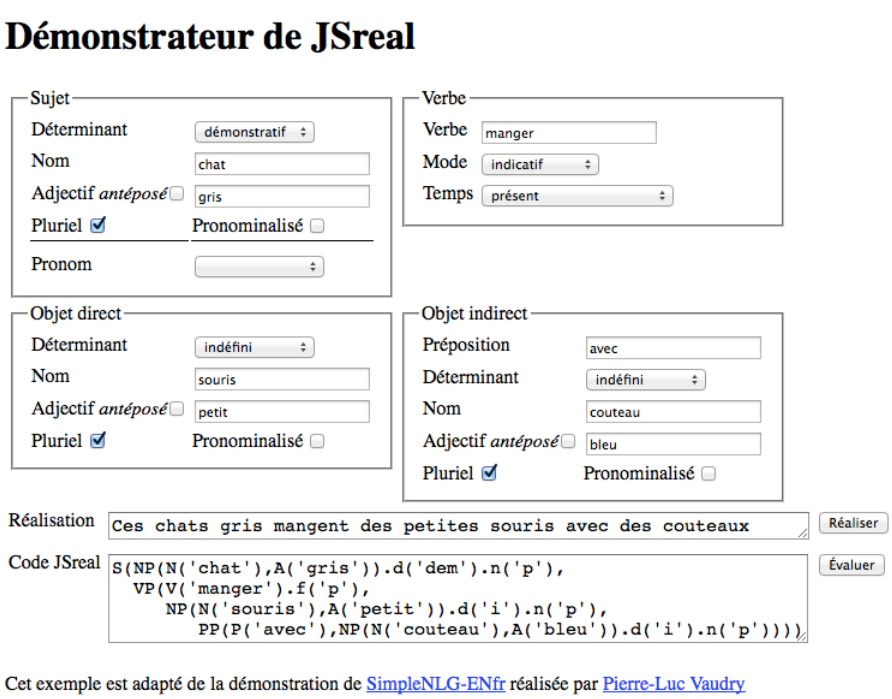

Fig. 1.8 Demo page to build a sentence with a subject (top left), a verb (top right), a direct object (bottom left) and indirect object (bottom right). For the subject, direct object and indirect object, a determiner, a noun and an adjective can be specified either with a text field or chosen from the menu. Once the fields are filled, clicking on the `Réaliser` button will fill the `Réalisation` text field with the French sentence and the other text field with the corresponding JSREAL code.

These few examples of use of JSREAL illustrate how it can be used to dynamically produce variants from a single input expression. Although they can be considered as toy examples, and they are, they comprise less than 100 lines of Javascript and a few lines of HTML and illustrate the potential of a dynamic text realizer within the browser.

1.7 Current limitations

Although JSREAL is already capable of complex and varied output, its grammatical coverage is relatively limited. In particular, it is missing some irregular verb forms and, because of time limits of the implementer, it does not yet support the specification of interrogative or negative clauses in a declarative way as it is possible in SIMPLENLG-ENFR.

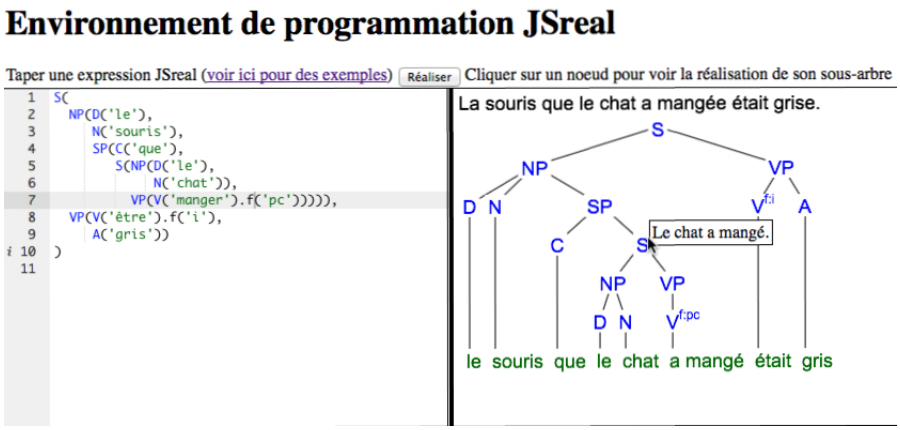

Fig. 1.9 JSREAL interactive development environment. The user enters a Javascript expression on the left; this expression is checked for some errors (e.g. mismatched parentheses or brackets) by the underlying ACE javascript editor. By clicking on the `Réaliser` button, the user can see on the right the realization of the expression and the corresponding syntactic tree. Clicking on an internal node of the tree, a tool tip presents the realization of the subtree of this node. In this example, the full sentence reads as The mouse that the cat ate was grey in which the proper agreement is made between mouse, which is feminine in French, was made with the adjective grey and the past participle in the subordinate sentence; it is feminine because the direct object, `que` standing for the `souris`, appeared before in the sentence. Clicking on the internal `S` node, we get the realization of that sentence only in which the past participle is not changed because it has no direct object.

The current lexicon is another weakness, being adapted instead of made from the ground up. Some features are redundant with JSREAL's grammatical rules and while a limited base vocabulary is understandable considering the size limitations, the words could be selected according to their prevalence on the web.

Finally, JSREAL is currently French-only, a choice dictated by the linguistic expertise of the implementers. An English version would be much more useful, considering there are ten times as many English internet users as there are French [14]; a multilingual version would even be more useful.

1.8 Conclusion

We have described the design of JSREAL, a text realizer written in Javascript that can produce flexible French text output and we have given a few illustrative applications. JSREAL reaches its stated objectives: it can easily be integrated to a web page, as much as a tool to more easily obtain some expressions than as the backbone of complex, automatically generated documents from data extracted from databases or through AJAX calls to an information server. We now intend to experiment with JSREAL in the context of data-driven applications in a domain where data is constantly

Fig. 1.10 A classical French word play in action in the browser. Given a single JSREAL input specification, in which some words have been assigned classes (in the Javascript sense), it is possible to get many variants of a single text: with the menus the user can select the tense of the verbs (one out of four), the gender and the number of the nouns designating persons. The top figure shows the initial display and the bottom shows a display in which all nouns have been made plural. The nouns that have been changed by the menu are underlined to indicate the main changes, but note that verbs have also changed to agree with the now plural subjects. The fourth menu can be used to put emphasis (here done in italics) on a named entity of the text: the narrator shown above, the young man, the friend and the traveler. Both the underlined and the emphasis are indicated by CSS classes that have be assigned to the underlying DOM structure with appropriate JSREAL functions.

changing such as weather information and combine it with dynamic visualization. It could also be applied to other types of applications such as the verbalization of a personal agenda or to describe personal relations dynamically extracted from a network.

Compared to other realizers, JSREAL integrates little syntactic theory, rather making the user build individual units in a manner similar to a syntactic tree. We prefer to look at this apparent weakness as a strength: as long as the user knows how to group words in phrases and use JavaScript, he can have JSREAL output complex documents. In that respect, JSREAL is very similar to SIMPLENLG, whose specification are also built in its native programming language.

JSREAL being quite flexible, it could be used for teaching French in the context of an interactive language learning environment such as the DRILLTUTOR [12] in

which the designer of the drill specifies the goals of the sentences to teach. Currently the sentences are simple templates which limits the range of sentences that can be presented to the learner. JSREAL would allow a greater flexibility in the design of language drill sentences by enabling the designer of the drills to focus on the high level goals of the interactive generation process. This would create a more interesting and varied type of output for the user of the language tutor.

References

1. Daoust, N.: *Événements. Luc Lightbringer*.
   http://daou.st/lightbringer/index.php?ong=4. Visited 10 Jan 2014
2. Daoust, N.: *JSreal*.
   http://daou.st/JSreal. Visited 10 Sep 2013
3. Daoust, N.: *Démonstration. JSreal*.
   http://daou.st/JSreal/Demo. Visited 10 Sep 2013
4. *Echelle orthographique Dubois-Buyse. Ressources francophones de l'Education*.
   http://o.bacquet.free.fr/db2.htm. Visited 9 Sep 2013
5. Elhadad, M. *SURGE: A Syntactic Realization Grammar for Text Generation*. Computer Science — Ben Gurion University of the Negev.
   http://www.cs.bgu.ac.il/surge. Visited 9 Sep 2013
6. Gatt, A, Reiter, E.: *SimpleNLG: A realisation engine for practical applications*. Proceedings of the 12th European Workshop on Natural Language Generation, 90–93 (2009)
7. Lavoie, B., Rambow, Owen. *A Fast and Portable Realizer for Text Generation Systems*. Proceedings of the Fifth Conference on Applied Natural Language Processing, 265–268 1997
8. Portet F., Reiter E., Gatt A., Hunter J., Sripada S., Freer Y., Sykes C., *Automatic generation of textual summaries from neonatal intensive care data*, Artificial Intelligence, Volume 173, Issues 7–8, May 2009, Pages 789-816.
9. Reiter, E., Dale, R.: *Building Natural Language Generation Systems*. Cambridge University Press (2000)
10. Vaudry, P.-L.: *SimpleNLG-EnFr*. Département d'informatique et de recherche opérationnelle de l'Université de Montréal.
    http://www-etud.iro.umontreal.ca/v̈audrypl/snlgbil/snlgEnFr_english.html. Visited 10 Sep 2013
11. Vaudry, P.-L.; Lapalme, G.: *Adapting SimpleNLG for bilingual English–French realisation*. Proceedings of the 14th European Workshop on Natural Language Generation, 183–187 (2013)
12. Zock, M., and G. Lapalme, *A Generic Tool for Creating and Using Multilingual Phrasebooks*, NLPCS 2010 (Natural Language Processing and Cognitive Science), Funchal, Madeira - Portugal.
13. Downloadable NLG systems. Association for Computational Linguistics Wiki.
    http://aclweb.org/aclwiki/index.php?title=Downloadable_NLG_systems. Visited 9 Sep 2013
14. Internet World Users by Language. Internet World Stats.
    http://www.internetworldstats.com/stats7.htm. Visited 16 Jan 2014
15. KPML one-point access page. Universitat Bremen.
    http://www.fb10.uni-bremen.de/anglistik/langpro/kpml/README.html. Visited 9 Sep 2013
16. RealPro. CoGenTex, Inc.
    http://www.cogentex.com/technology/realpro/index.shtml. Visited 10 Sep 2013
17. SimpleNLG.
    http://code.google.com/p/simplenlg. Visited 9 Sep 2013

18. Table of NLG systems. In: NLG Systems Wiki.
http://www.nlg-wiki.org/systems/Table_of_NLG_systems. Visited 9 Sep 2013