TITUS E. C. WORMER

RETEXT

RETEXT

Design of an extensible system for analysing and manipulating natural language

TITUS E. C. WORMER

School of Design and Communication
Communication and Multimedia Design
Amsterdam University of Applied Sciences

August 2014 – version 0.5

Titus E. C. Wormer (student № 500625392): *Retext,* Design of an extensible system for analysing and manipulating natural language, © August 2014

SUPERVISOR: Justus Sturkenboom

SUBMISSION: August 18, 2014

LOCATION: Delft

EMAIL: tituswormer@gmail.com

ACKNOWLEDGMENTS & DEDICATION

Thanks to my supervisor Justus Sturkenboom for the trust (and the patience) in me and my work, and for allowing me to produce a product I'm pleased with.

In addition, thanks go out to the open source community, especially those who raised issues, submitted pull request, and those who haven't done so yet, but will in the future.

This thesis is dedicated to Jelmer, who departed this happy live too early.

ABSTRACT

This document captures the use cases and requirements for designing and standardising a solution for textual manipulation and analysis in ECMAScript. In addition, this paper presents an implementation that meets these requirements and answers these use cases.

EXECUTIVE SUMMARY

Natural Language Processing (NLP) covers many different tasks, but the process of accomplishing these goals touches on well defined stages (§ 1.1, p. 3). Such as *tokenisation*, the main focus of the proposal. Current implementations on the web platform are lacking (§ 1.3, p. 4). In part, because advanced machine learning techniques (such as *supervised learning*) do not work on the web (§ 1.3.3, p. 7).

The audience that benefits the most from better parsing on the web platform, are web developers, a group which is more interested in practical use, and less so in theoretical applications (§ 2.1, p. 9).

The target audiences use cases for NLP on the web are vast. Examples include automatic summarisation, sentiment recognition, spam detection, typographic enhancements, counting words, language recognition, and more (§ 2.2, p. 9).

The presented proposal is split into several smaller solutions. These solutions come together in a proposal: *Retext*, a complete natural language system (§ 3, p. 15). *Retext* takes care of parsing natural language and enables users to create and use plugins (§ 3.4, p. 18).

Parsing is delegated to *parse-latin* and others, which first tokenise text into a list of words, punctuation, and white space. Later, these tokens are parsed into a syntax tree, containing paragraphs, sentences, embedded content, and more. Their intelect extends several well known techniques (§ 3.2, p. 16).

The objects returned by *parse-latin* and others are defined by NLCST. NLCST defines the syntax for these objects. NLCST is designed in simmilarity to other popular syntax tree specifications (§ 3.1, p. 15).

The interface to analyse and manipulate these object is implemented by TextOM. TextOM is created in similarity to other, for the target audience well known, techniques (§ 3.3, p. 17).

The proposal was validated both by solving the audiences use cases, and by measuring the audiences enthusiasm. The use cases were validated by implementing many as plugins for *Retext* (§ 4.1, p. 19). The enthusiasm showed by the target audience on social networks, through email, and social coding was positive (§ 4.2, p. 19).

INTRODUCTION

NLP, a field of computer science, artificial intelligence, and linguistics concerned with the interaction between computers and human languages, is becoming more important in society. For example, search engines provide answers before being questioned, intelligence agencies detects threats of violence in text messages, and e-mail applications know if you forgot to include an attachment.

Despite increased interest, web developers trying to solve NLP problems reinvent the wheel over and over again. There are tools, especially for other platforms—such as in Python (Bird et al.) and Java (Baldridge)—but they either take a too naive approach[1], or try to do everything out of the box[2]. What is missing is a standard representation of the grammatical hierarchy of text and a standard for multipurpose analysis of natural language.

My initial motivation for natural language was sparked by typography, when I felt the need to create a typographically beautiful blog, somewhere in the summer of 2013. I felt a craving to apply the tried-and-true practices of typography found on paper, to the web. I was inspired by how these practices were available on other platforms, on TEXor LATEX, with tools such as *microtype* (Schlicht), and the *ClassicThesis* theme (Miede) based on *The Elements of Typographic Style* (Bringhurst).

My interest for fixing typography on the web was piqued. Thus, I began work on *MicroType.js*, an unpublished library, to facilitate several graphic and typographic practices on the web. Examples include automatic initials, ligatures, optical margin alignment, initial recognition, smart punctuation, automatic hyphenation, character transpositions, and more. The possibilities were vast, but I noticed how the underlying parser and data representation, the definition of words, white space, punctuation, and sentences, were not good enough. The blog never came into existence, but during this thesis, I had the chance to finally fix this problem. While working on this thesis, the specification, the product, I developed a well thought out and substantiated solution.

Retext—the implementation introduced in this thesis—and other projects in the Retext family are a new approach to the syntax of natural text. Together they form an extensible system for multipurpose analysis of natural language in ECMAScript.

To achieve this goal, I have organised my paper into five main chapters. In the first chapter I define the scope of this paper and review

1 Such as ignoring white space (Loadfive), implementing a naive definition of "words" (Hunzaker), or by using an inadequate algorithm to detect sentences (New York Times).

2 Although a do-all library works well on server-side platforms, it fares less well on the web (such as Umbel et al.), where modularity and moderation are in order.

current implementations, what they lack and where they excel. In the second chapter, I propose a better implementation and show its architectural design. In the third chapter, I define conditions for the production of such a proposal, where I touch upon the target audience, use cases, and requirements.

I end the paper with a fourth chapter which describes the steps taken to validate the proposal. I conclude with a fifth chapter that offers information on expanding the proposal.

Before I can begin the examination of a proposal, I need to provide a context for NLP.

CONTENTS

Part I

RETEXT

1

CONTEXT

1.1 NATURAL LANGUAGE PROCESSING

> *Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.*
>
> — Elizabeth D. Liddy ('Natural language processing')

The focus of this paper is Natural Language Processing (NLP). NLP is a field related to human–computer interaction, as it concerns itself with enabling machines to understand human language. Human language, a medium which is easy for humans to understand, poses problems for machines. The Georgetown–IBM experiment in 1945, one of the first application of NLP, illustrates this difficulty. During this study in New York, scientists demonstrated a Russian–English translation system (Hutchins). The machine translated more than sixty sentences from Russian to English. The experiment was well publicised and resulted in optimism. The public believed machine-translation would be a "solved problem" within three to five years. Despite promising initial results, the following ten years proved to be disappointing and led to reduced funding.

Machine translation is just one of many major tasks involved with NLP. Other tasks include generating summaries, detecting references to people and places, or extracting opinion. Tasks which are all part of *information extraction*: the act of finding certain information in a document. Many programs exists to carry out these and many other NLP tasks. The approach taken to perform these tasks are often similar between implementations. Entity linking for example, is often implemented as follows (according to 'Stanbol'):

1. *Language Detection (optional)* — Based on the language of the given text, the algorithms behind the following steps will change. Omitted if the implementation supports a single language;
2. *Sentence Tokenisation (optional)* — Sentence breaking elevates performance and heightens accuracy of the following stages, in particular POS tagging;
3. *Word Tokenisation* — The entities (words) must be free from their surroundings;

4. *Part-of-Speech (POS) Tagging (optional)* — It is often desired to link several nouns or proper nouns. Detecting word categories makes this achievable;
5. *Noun Phrase Detection (optional)* — Although *apple* and *juice* could be two entities, it is more appropriate to link to one entity: *apple juice*. Detecting noun phrases makes this possible;
6. *Lemmatisation or Stemming (optional)* — Be it *walk*, *walked*, or *walking*, all forms of *walk* could link to the same entity. Detecting either makes this possible;
7. *Entity Linking* — Linking detected entities to references, such as an encyclopaedia.

NLP covers many different tasks, but the process of accomplishing these goals touches, as seen above, on well defined stages.

1.2 SCOPE

Although many NLP tasks exist, the standard and the implementation this paper proposes will only cover one: *tokenisation*. Tokenisation, as defined here, includes breaking sentences, words, and other grammatical units.

Another confinement set to scope the proposal, is that it focusses on syntactic grammatical units. Thus, semantic units (i.e., phrases and clauses) are ignored.

In addition, the paper focusses on written language (text), thus ignoring spoken language.

Last, this paper focusses on Latin script languages: written languages using an alphabet based on the the classical Latin alphabet.

1.3 IMPLEMENTATIONS

While researching algorithms to tokenise natural language few viable implementations were found. Most algorithms look at either sentence- or word tokenisation (rarely both). This section describes the current implementations, where they excel, and what they lack.

1.3.1 *Stages*

This section delves into how current implementations accomplish tokenisation tasks.

1.3.1.1 *Sentence tokenisation*

Often referred to as sentence boundary disambiguation[3], sentence tokenisation is an elementary but important part of NLP. It is almost always a stage in NLP applications and not an end goal. Sentence tokenisation makes other stages (e.g., detecting plagiarism or Part-of-Speech (POS) tagging) perform better.

Oftentimes, sentences end in one of three symbols: either a full stop (.), an interrogative point (?), or an exclamation point (!)[4]. But detecting the boundary of a sentence is not as simple as breaking it at these markers: they might serve other purposes. Full stops often occur in numbers, suffixed to abbreviations or titles, in initialisms[5], or in embedded content[6]. The interrogative- and exclamation points too can occur ambiguously, such as in a quote (e.g., "'Of course!", she screamed').

Disambiguation gets even harder when these exceptions *are* in fact a sentence boundary (double negative), such as in "…use the feminine form of idem, ead." or in "'Of course!", she screamed, "I'll do it!"', where in both cases the last terminal marker ends the respective sentence.

1.3.1.2 *Word tokenisation*

Like sentence tokenisation, word tokenisation is another elementary but important stage in NLP applications. Whether stemming, finding phonetics, or POS tagging, tokenising words is an important precursory step.

Often implementations see words as everything that is *not* white space (i.e., spaces, tabs, feeds) and their boundaries as everything that is (Loadfive).

Some implementations take punctuation marks into account as boundaries. This practice has flaws, as it results in the faulty classification of inter-word punctation[7] as part of the surrounding word (Umbel et al.).

---

3 Both sentence tokenisation and sentence boundary disambiguation detect sentences. Sentence boundary disambiguation focusses on the position where sentences break (as in, "One sentence?| Two sentences.|", where the pipe symbols refer to the end of one sentence and the beginning of another), whereas sentence tokenisation targets both the start and end location (as in, "{One sentence?} {Two sentences.}", where everything between braces is classified as a sentence).

4 One could argue the in 1962 introduced obscure interrobang (‽), used to punctuate rhetorical statements where neither the question nor exclamation alone exactly serve the writer well, should be in this list (Spector).

5 Although the definition of initialism is ambiguous, this paper defines its use as an acronym (an abbreviation formed from initial components, such as "sonar" or "FBI") with full stops depicting elision (such as "e.g.", or "K.G.B.").

6 Embedded content in this paper refers to an external (non-grammatical) value embedded into a grammatical unit, such as a hyperlink or an emoticon. Note that these embedded values often consist of valid words and punctuation marks, but most always shouldn't be classified as such.

7 Many such inter-word symbols exist, such as hyphenation points, colons ("12:00"), or elision (whether denoted by full stops, "e.g."; apostrophes, the Dutch "'s"; or slashes, "N/A").

1.3.2 *Tasks*

The previous section covered implementations that solve tokenisation stages in NLP applications, such as Natural's word tokenisers (Umbel et al.). Concluded was that these implementations are lacking. This section covers several implementations that solve these stages as part of a larger task.

1.3.2.1 *Sentiment Analysis*

Sentiment analysis is an NLP task concerned with the polarity (positive, negative) and subjectivity (objective, subjective) of text. It could be part of an implementation to detect messages with a certain polarity. Twitter allows its users to search on polarity. For example, when a user searches for "movie :)", Twitter searches for positive tweets.

The implementation of sentiment analysis could look as follows:

1. *Detect Language (optional)*;
2. *Sentence Tokenisation (optional)* — Different sentences have different sentiments, tokenising them helps provide better results;
3. *Word Tokenisation* — Needed to compare with the database;
4. *Lemmatisation or Stemming (optional)* — Helps classification;
5. *Sentiment Analysis*.

Sentiment analysers typically include a database mapping either words, stems, or lemmas to their respective polarity and/or subjectivity[8] and return the average sentiment per sentence, or for the whole document.

In the case of the previously mentioned Twitter example, the service filters out all neutral and negative results, and return the remaining (positive attitude) results.

Many implementations exist for this task (Roth; Zimmerman; Sliwinski), many of which do not include inter-word punctuation in their *definition* of words, resulting in less than perfect results[9].

1.3.2.2 *Automatic Summarisation*

Automatic summarisation is an NLP task concerned with the reduction of text to the *major* points retaining the original document. Few open source implementations of automatic summarisation algorithms on the web, in contrast with implementations for sentimental analysis, were found[10]. The implementation of automatic summarisation could look as follows:

8 For example, the AFINN database mapping words to polarity (Nielsen).

9 In fact, all found implementations deploy lacking tokenisations steps. Dubious, as they each create unreachable code through their naivety: all implementations remove dashes from words, while words such as "self-deluded" are included in the databases they use, but never reachable.

10 For example, on the web only node-summary was found (Brooks), in Scala textteaser was found (Balbin, 'textteaser').

1. *Detect Language (optional)*;
2. *Sentence Tokenisation (optional)* — Unless even finer grained control over the document is possible (tokenising phrases), sentences are the smallest unit that should stay intact in the resulting summary;
3. *Word Tokenisation* — Needed to calculate keywords (words which occur more often than expected by chance alone);
4. *Automatic summarisation.*

Automatic summarisers typically return the highest ranking units, be it sentences or phrases, according to several factors:

A. *Number of words* — An ideal sentence is neither too long nor too short;
B. *Number of keywords* — Words which occur more often than expected by chance alone in the whole text;
C. *Similarity to title* — Number of words from the document's title the unit contains;
D. *Position inside parent* — Initial and final sentences of a paragraph are often more important than sentences buried somewhere in the middle.

Some implementations include only keyword metrics (Brooks), others include all features (Balbin, 'textteaser'), or even more advanced factors ('Summly').

The only implementation working on the web, by James Brook ('node-summary'), takes a naive sentence tokenisation approach. Such as ignoring sentences terminated by exclamation marks. Both other implementations, and many more, use a whole different approach to sentence tokenisation: Corpora.

1.3.3 *Using Corpora for NLP*

A corpus is a large, structured set of texts used for many NLP and linguistics tasks. Corpora contain items (often words, but sometimes other units) annotated with information (such as POS tags or lemmas).

These colossal (often more than a million words[11]) lumps of data are the basis of many of the newer revolutions in NLP (Mitkov et al.). Parsing based on supervised learning (in NLP, based on annotated corpora), is the opposite of rule based parsing[12]. Instead of rules (and exceptions to these rules, exceptions to these exceptions, and so on) specified by a developer, supervised learning[13] delegates this task

11 The Brown Corpus contains about a million words (Francis and Kučera), the Google N-Gram Corpus contains 155 billion (Brants and Franz).

12 A simple rule based sentence tokeniser could be implemented as follows (O'Neil):

A. If it is a period, it ends a sentence;
B. If the period is preceded by an abbreviation, it does not end a sentence;
C. If the next token is capitalised, it ends a sentence.

13 "[From] a set of labeled examples as training data … [, make] predictions for all unseen points" (Mohri et al.).

to machines. This delegation results in a more performant, scalable, program.

Parsing based on corpora has proven to be better in several ways over rule based parsing, but the former has disadvantages:

1. Good training sets are required;
2. If the corpus was created from news articles, algorithms based on it will not fair so well on microblogs (e.g., Twitter posts).
3. Some rule based approaches for pre- and post processing are still required;

In addition, corpora based parsing will not work well on the web. Loading corpora over the network each time a user request a web page is unfeasible for most web sites and applications[14].

Two viable alternative approaches exist for the web: rule based tokenisation, or connecting to a server over the network.

1.3.4 *Using a web API*

Whereas the term Application Programming Interface (API) stands for an interface between two programs, it is often used in web development as requests (from a web browser), and responses (from a web server) over Hypertext Transfer Protocol (HTTP). For example, Twitter has such a service to allow developers to list, replace, create, and delete so-called tweets and other objects (users, images, &c.). This paper uses the term Web API for the latter, and API for any programming interface.

With the rise of the asynchronous web[15], supervised learning became available through web APIs (Balbin, 'TextTeaser'; Princeton University; 'TextRazor'). This made it possible to use supervised learning techniques on the web, without needing to download corpora to a users computer.

However, accessing NLP web APIs over a network has disadvantages. Foremost of which the time involved in sending data over a network and bandwidth used (especially on mobile networks), and heightened security risks.

14 Currently, one technology exists for storing large datasets in a browser: the HTML5 File System API. However, "work on this document has been discontinued", and the specification "should not be used as a basis for implementation" (Uhrhane).

15 Starting around 2000, Asynchronous JavaScript and XML (AJAX) started to transform the web. Beforehand, significant changes to websites only occurred when a user navigated to a new page. With AJAX however, new content arrived to users without the need for a full page refresh. One of the first examples are the Outlook Web App in 2000 ('Outlook Web Access - A Catalyst for Web Evolution') and Gmail in 2004 (Hyder), both examples of how AJAX made the web feel more "app-like".

2

PRODUCTION

2.1 TARGET AUDIENCE

The audience that benefits the most from the proposal, are web developers. Web developers are programmers who specialise in creating software that functions on the world wide web. A group which enables machines to respond to humans. They engage in client side development (building the interface between a human and a machine on the web), and sometimes also in server side development (building the interface between the client side and a server).

Typical areas of work consist of programming in ECMAScript, marking up documents in Hypertext Markup Language (HTML), graphic design through Cascading Style Sheets (CSS), creating a back end in Node.js, PHP: Hypertext Preprocessor (PHP), or other platforms, contacting a MongoDB, MySQL, or other database, and more.

Additionally, many interdisciplinary skills, such as usability, accessibility, copywriting, information architecture, or optimisation, are also of concern to web developers.

2.2 USE CASES

The use cases of the target audience, the web developer, in the field of NLP are many. Research for this paper found several use cases, although it is expected many more could be defined. The tasks below are each categorised into broad, generic fields: analysation, manipulation, and creation.

A. The developer may intent to summarise natural text (mostly analysation, potentially also manipulation);
B. The developer may intent to create natural language, e.g., displaying the number of unread messages: "You have 1 unread message," or "You have 0 unread messages" (creation);
C. The developer may intent to recognise sentiment in text: is a *tweet* positive, negative, or spam? (analysation);
D. The developer may intent to replace so-called *dumb* punctuation with *smart* punctuation, such as dumb quotations with (") or ("), three dots with an ellipsis (...), or two hyphens with an en-dash (–) (manipulation);

E. The developer may intent to count the number of certain grammatical units in a document, such as, words, white space, punctuation, sentences, or paragraphs (analysation);
F. The developer may intent to recognise the language in which a document is written (analysation);
G. The developer may intent to find words in a document based on a search term, with regards for the lemma (or stem) and/or phonetics (so that a search for "smit" also returns similar words, such as "Schmidt" or "Smith") (analysation and manipulation).

NLP is a large field with many challenges, but not every challenge in the field is of interest to the web developer. Foremost, the more academic areas of NLP, such as speech recognition, optical character recognition, text-to-speech transformation, translation, and machine learning, do not fit well with the goals of web developers.

2.3 REQUIREMENTS

The proposal must enable the target audience to reach the in the previous section defined use cases. In addition, the proposal should meet several other requirements to better suit the wishes of the target audience.

2.3.1 *Open Source*

To reach the target audience and validate its usability, the proposal should be open source. All code should be licensed under MIT, a license which provides rights for others to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the code it covers.

In addition, the software should be developed under the all-seeing eye of the community: on GitHub. GitHub is a hosted version control[16] service with social networking features. On GitHub, web developers follow their peers to track what they are working on, watch their favourite projects to get notified of changes, and raise issues and feature requests.

2.3.2 *Performance*

The proposal should execute at high *performance*. Performance includes the software having a small file size to reach the client over the network with the highest possible speed, but most importantly that the execution of code should operate efficiently and at at high speeds.

16 Version control services manage revisions to documents, popularly used for controlling and tracking changes in software.

2.3.3 *Testing*

Testing should have high priority in the development of the proposal. Testing, in software development, refers to validating if software does what it is supposed to do, and can be divided into several subgroups:

A. *Unit testing* — Validation of each specific section of code;
B. *Integration testing* — Validation of how programs work together;
C. *System testing* — Validation of if the system meets its requirements;
D. *Acceptance testing* — Validation of the end product.

Great care should be given to develop an adequate test suite with full *coverage* for every program. Coverage, in software development, is a term used to describe the amount of code tested by the test suite. Full coverage means every part of the code is reached by the tests.

Unit test run through Mocha (Holowaychuk), coverage is detected by Istanbul (Anantheswaran).

2.3.4 *Code quality*

Code quality—how useful and readable for both humans and machines the software is—should be vital. For humans, the code should be consistent and clear. For computers, the code should be free of bugs and other suspicious code.

2.3.4.1 *Suspicious Code and Bugs*

To detect bugs and suspicious code in the software, Eslint is used (Zakas). *Linting*, in computer programming, is a term used to describe static code analysis to detect syntactic discrepancies without running the code. Eslint is used because it provides a solid set of basic rules and enables developers to create custom rules.

2.3.4.2 *Style*

To enforce a consistent code style—to create readable software for humans—JSCS is used (Dulin). JSCS provides rules for (dis)allowing certain code patterns, such as white space at the end of a line or camel cased variable names, or enforcing a maximum line length. JSCS was chosen because it, just like Eslint, provides a strong basic set of rules. The rules chosen for the development of the proposal were set strict to enforce all code to be written in the same manner.

2.3.4.3 *Commenting*

Even when code is bug free, uses no confusing shortcuts, and adheres to a strict style, it might still be hard to understand for humans. *Commenting* code—describing what a program does and why it accomplishes this in a certain way—is important. However, commenting can also be

too verbose, for example when the code is duplicated in natural language.

JSDoc ('Annotating JavaScript for the Closure Compiler') is a markup language for ECMAScript that allows developers to embed documentation—using comments—in source code. Several tools can later extract this information and expose it separate from the original code. "Tricky" code should be annotated inside the software with comments to help readers understand why certain decisions were made.

2.3.5 *Automation*

When suspicious, ambiguous, or buggy code is introduced in the software, the error should be automatically detected. In some cases deployment should be prevented. Automated Continuous Integration (CI) environments to enforce error detection should be used. To detect complex, duplicate, or bug-prone code, Code Climate is used ('Code Climate'). To validate all tests passed before deploying the software, Travis is used ('Travis').

2.3.6 *API Design*

Quality interface design should have high priority for the proposal. A good API, according to Joshua Bloch ('How to design a good API and why it matters'), has the following characteristics:

1. Easy to learn;
2. Easy to use;
3. Hard to misuse;
4. Easy to read;
5. Easy to maintain;
6. Easy to extend;
7. Meeting its requirements;
8. Appropriate for the target audience.

In essence equal, but worded differently, are the characteristics of good API design according to the Qt Project ('API Design Principles'):

1. Be minimal;
2. Be complete;
3. Have clear and simple semantics;
4. Be intuitive;
5. Be easy to memorise;
6. Lead to readable code.

The proposal should take these characteristics, and the in their sources given examples, into account.

2.3.7 *Installation*

Simple access to the software for the target audience, both on the client side and on the server side, should be given high priority. On the client side, many different package managers exist, the most popular being Bower and Component[17]. For Node.js (on the server side), npm is the most popular. To reach the target audience, besides making the source available on GitHub, all popular package managers, NPM, Bower, and Component, are used.

17 Popularity here is simply defined as having the highest number of search results on Google.

3

DESIGN & ARCHITECTURE

The in this paper presented solution to the problem of NLP on the client side is split up in multiple small proposals. Each sub-proposal solves a subproblem.

A. *NLCST* — Defines a standard for classifying grammatical units understandable for machines;
B. *parse-latin* — Classifies natural language according to Natural Language Concrete Syntax Tree (NLCST);
C. *TextOM* — Provides an interface for analysing and manipulating output provided by parse-latin;
D. *Retext* — Provides an interface for transforming natural language into an object model and exposes an interface for plugins.

The decoupled approach taken by the provided solution enables other developers to implement their own software to replace sub-proposals. For example, other parties could create a parser for the Chinese language and use it instead of parse-latin to classify natural language according to NLCST, or other parties can implement an interface like TextOM with functionality for phrases and clauses.

3.1 SYNTAX: NLCST

To develop natural language tools in ECMAScript, an intermediate representation of natural language is useful. Instead of each module (for example, every stage in section 1.3.2.1 on page 25) defining its own representation of text, using a single syntax leads to better interoperability, performance, and results.

The elements defined by Natural Language Concrete Syntax Tree (NLCST) are based on the the grammatical hierarchy, but by default do not expose all its constituents[18]. Additionally, NLCST provides more elements to cover other semantic units in natural language[19].

The definitions were influenced by other syntax trees specifications for manipulation on the web platform, such as *CSS*, eponymous for the CSS language (Holowaychuk et al., 'css') or the *Mozilla JavaScript AST*, for ECMAScript ('Parser API').

18 The grammatical hierarchy of text is constituted by words, phrases, clauses, and sentences. NLCSTs only implements the sentence and word constituents by default, although clauses and phrases could be provided by implementations.

19 Most notably, punctuation, symbol, and white space elements.

Both widely used implementations, *CSS* by Rework (Holowaychuk et al., 'rework'), and *Mozilla JavaScript AST* by Esprima (Hidayat), Acorn (Haverbeke), and Escodegen (Suzuki).

NLCST is different from both these specifications, as it implements a Concrete Syntax Tree (CST), wheres the others use an Abstract Syntax Tree (AST). A CST is a one-to-one mapping of source (such as natural language) to result (a tree). All information stored in the source is also available through the tree (Bendersky). This makes it easy for developers to save the output or pass it on to other libraries for further processing. However, the information stored in CSTs is verbose, which might be difficult to work with.

See appendix A on page 25 for a complete list of specified nodes of NLCST.

3.2 PARSER: PARSE-LATIN

To create a syntax tree according to NLCST from natural language, this paper presents *parse-latin* for Latin script based languages[20]. Additionally, to prove the concept, two other libraries are presented, *parse-english* and *parse-dutch*. Both with parse-latin as a basis, but providing better support for several language specific features, respectively for English and Dutch.

By using the CST as described by NLCST and the *parse-latin* parser, the intermediate representation can be used by developers to create independent modules which may receive better results or performance over implementing their own parsing tools.

In essence, *parse-latin* tokenises text into white space, word, and punctuation tokens. *parse-latin* starts out with a pretty simple definition, one that some other tokenisers also implement (MacIntyre):

1. A *word* is one or more, letter or number characters;
2. A *white space* is one or more white space characters;
3. A *punctuation* is one or more of anything else.

Then, *parse-latin* manipulates and merges those tokens into a syntax tree, adding sentences, paragraphs, and other nodes where needed. The majority of the intelect of the algorithm deals with sentence tokenisation (§ 1.3.1.1, p. 5). This is done in similar fashion, but more intelligent, to *Emphasis* (New York Times).

A. *Inter-word punctuation* — Some punctuation marks are part of the word they occur in, such as the punctuation marks in "non-profit", "she's", "G.I.", "11:00", or "N/A";
B. *Non-terminal full-stops* — Some full-stops do not mark a sentence end, such as the full stops in "1.", "e.g.", or "id.";
C. *Terminal punctuation* — Although full-stops, question marks, and exclamation marks (sometimes) end a sentence, that end might

20 Such as Old-English, Icelandic, French, or even scripts slightly similar, such as Cyrillic, Georgian, or Armenian.

not occur directly after the mark, such as the punctuation marks following the full stop in ".)" or "."";
D. *Embedded content* — Punctuation marks are sometimes used in non-standard ways, such as when a section or chapter delimiter is created with a line containing three asterisk marks ("* * *").

See appendix B on page 29 for example output provided by *parse-latin*.

3.2.1 *parse-english*

parse-english provides the same interface as *parse-latin*, but returns results better suited for English text. Exceptions in the English language include:

A. *Unit abbreviations* — "tsp.", "tbsp.", "oz.", "ft.", &c.;
B. *Time references* — "sec.", "min.", "tues.", "thu.", "feb.", &c.;
C. *Business Abbreviations* — "Inc." and "Ltd."
D. *Social titles* — "Mr.", "Mmes.", "Sr.", &c.;
E. *Rank and academic titles* — "Dr.", "Gen.", "Prof.", "Pres.", &c.);
F. *Geographical abbreviations* — "Ave.", "Blvd.", "Ft.", "Hwy.", &c.;
G. *American state abbreviations* — "Ala.", "Minn.", "La.", "Tex.", &c.;
H. *Canadian province abbreviations* — "Alta.", "Qué.", "Yuk.", &c.;
I. *English county abbreviations* — "Beds.", "Leics.", "Shrops.", &c.;
J. *Elision (omission of letters)* — "'n' ", "'o", "'em", "'twas", "'80s", &c.

3.2.2 *parse-dutch*

parse-dutch has, just like *parse-english*, the same interface as *parse-latin*, but returns results better suited for Dutch text. Exceptions in the Dutch language include:

A. *Unit and time abbreviations* — "gr.", "sec.", "min.", "ma.", "vr.", "vrij.", "febr", "mrt", &c.;
B. *Many other common abbreviations* — "Mr.", "Mv.", "Sr.", "Em.", "bijv.", "zgn.", "amb.", &c.;
C. *Elision (omission of letters)* — "d"', "'n", "'ns", "'t", "'s", "'er", "'em", "'ie", &c..

3.3 OBJECT MODEL: TEXTOM

To modify an NLCST tree in ECMAScript, whether created by *parse-latin*, *parse-english*, *parse-dutch*, or other parsers, this paper presents TextOM. TextOM implements the nodes defined by NLCST, but provides an object-oriented style[21] to manipulate these nodes. TextOM was designed to be similar to the Document Object Model (DOM)[22], the

21 Object-oriented programming is a style of programming, where classes, instances, attributes, and methods are important.
22 See appendix C on page 31 for more information on the DOM.

mechanism used by browsers to expose HTML through ECMAScript to developers. Because of TextOMs likeness to the DOM, TextOM is easy to learn and familiar to the target audience.

TextOM provides functionality for events (a mechanism for detecting changes), modification (inserting, removing, and replacing children into/from parents), and traversal (such as finding all words in a sentence).

NLCST allows authors to extend the specification by defining their own units, for example creating phrase or clause nodes. TextOM allows for the same extension, and is build to work well with these "unknown" nodes.

3.4 NATURAL LANGUAGE SYSTEM: RETEXT

For natural language processing on the client side, this paper presents *Retext*. *Retext* combines a parser, such as *parse-latin* or *parse-english*, with an object model: TextOM. Additionally, *Retext* provides a minimalistic plugin mechanism which enables developers to create and publish plugins for others to use, and in turn enables them to use others' plugins inside their projects.

Retext provides a strong basis to use plugins in order to add simple natural language features to a website, but additionally provides functionality to extend this basis—create plugins, parsers, or other features—to create vast natural language systems.

4

VALIDATION

The presented proposal was validated through two approaches. The design and the usability of the interface was validated through solving several use cases of the target audience with the proposal. Interest in the proposal by the target audience was validated by measuring the enthusiasm showed by the open source community.

4.1 PLUGINS

More than fifteen plugins for *Retext* were created to confirm if, and validate how, the proposals integrated together, and how the system worked as a whole. Including tools for:

A. Transforming so-called dumb punctuation marks into more typographically correct punctuation marks (‘retext-smartypants’);
B. Transforming emoji short-codes (:cat:) into real emoji (‘retext-emoji’);
C. Detecting the direction of text (‘retext-directionality’);
D. Detecting phonetics (‘retext-double-metaphone’);
E. Detecting the stem of words (‘retext-porter-stemmer’);
F. Detecting grammatical units (‘retext-visit’);
G. Finding text, even misspelled (‘retext-search’);
H. Detecting POS tags (‘retext-pos’);
I. Finding keywords and -phrases (‘retext-keywords’).

During the development of these plugins, several problems were brought to light in the developed software. These problems were recursively dealt with, back and forth, between the software and the plugins. The software changed severely by these changes, which resulted in a better interface and usability.

4.2 RECEPTION

To confirm interest by the target audience in the proposal, enthusiasm showed by the open source community was measured. To initially spark interest, several blogs and email newsletters were contacted to feature *Retext*, either in the form of an article or as a simple link. This resulted in coverage on high-profile blogs (Young) and newsletters (Cooper, ‘Issue 47: August 7, 2014’; ‘Issue 193: August 8, 2014’; Newspaper.io). Later, organical growth resulted in features

on link roundups (Misiti; Sorhus), reddit (Polencic; Wormer, 'Natural Language Parsing with Retext'; 'DailyJS: Natural Language Parsing with Retext').

In turn, these publications resulted in positive reactions, such as on Twitter (dailyjs; Ahmed; Oswald; Rinaldi; Grigorik; JavaScript Daily), other websites, and both feedback and fixes on GitHub (gut4; Gonzaga dos Santos Filho; rbakhshi; Burkhead).

Additionally, many of the target audience started following the project on GitHub ('Stargazers').

5

CONCLUSION

This chapter consists of a short summary (§ 5.1), a list of conclusions (§ 5.2), and suggestions for future work (§ 5.3).

5.1 SUMMARY

NLP covers many different tasks, but the process of accomplishing these goals touches on well defined stages. Such as *tokenisation*, the main focus of this paper. Current implementations on the web platform are lacking. In part, because techniques such as *supervised learning* do not work on the web.

The audience that benefits the most from better parsing on the web platform, are web developers, a group which is more interested in practical use, and less so in theoretical applications.

The presented proposal is split up in several solutions: a specification, a parser, and an object model. These solutions come together in a proposal: *Retext*, a complete natural language system. *Retext* takes care of parsing natural language and enables users to create and use plugins.

The proposal was validated both by solving the audiences use cases, and by measuring the audiences enthusiasm.

5.2 CONCLUSIONS

5.3 LIMITATIONS & FUTURE WORK

The proposal leaves open many areas of interest for future investigation. Some of which, are featured here.

A. *Internationalisation* — Currently, the proposal is only tested on Latin script languages. The software was developed with other languages and scripts, such as Arabic, Hangul, Hebrew, and Kanji, in mind. Future work could expand support to include these scripts;
B. *Difference application* — Currently, the proposal does not support difference application. When a word is added at the end of a sentence, all steps to produce the output have to be revisited. Although the proposal is created with this in mind, no support

has been added. Future work could include difference application support;

C. *Non-rule based parsing* — Currently, NLCST trees are created with rule based parsers. But, corpora based parsers could also produce these trees. Future work could investigate and implement such supervised-learning approaches.

D. *Academic goals* — Currently, the proposals cater to practical use cases. Future work could expand on this purview and implement more academic goals.

E. *Semantical units* — Currently, the proposals provide syntactic units. Future work could expand on this by providing information about phrases and clauses to users.

F. *Import formants* — Currently, the parsers each require plain text input. Future work could expand on this by allowing other input formats, such as MarkDown or LaTeX.

G. *Heighten Performance* — The decision made to adopt an object-oriented approach for analysation and manipulation, came at a huge performance cut. When implementing both TextOM and *parse-latin* over just *parse-latin*, performance decreases over 90%. Future work should investigate and implement better performance.

Part II

APPENDIX

A

NLCST DEFINITION

NODE

Node represents any actual unit in the NLCST hierarchy.

```
interface Node {
    type: string;
}
```

PARENT

Parent (Node) represents a unit in the NLCST hierarchy which can have zero or more children.

```
interface Parent <: Node {
    children: [];
}
```

TEXT

Text (Node) represents a unit in the NLCST hierarchy which has a value.

```
interface Text <: Node {
    value: string | null;
    location: Location | null;
}
```

LOCATION

Location represents the node's actual location in the source input.

```
interface Location {
    start: Position;
    end: Position;
}
```

POSITION

Position represents the a position in the source input.

```
interface Position {
    line: uint32 >= 1;
    column: uint32 >= 1;
}
```

RootNode

Root (Parent) represents the document.

```
interface RootNode < Parent {
    type: "RootNode";
}
```

ParagraphNode

Paragraph (Parent) represents a self-contained unit of a discourse in writing dealing with a particular point or idea.

```
interface ParagraphNode < Parent {
    type: "ParagraphNode";
}
```

SentenceNode

Sentence (Parent) represents a grouping of grammatically linked words, that in principle tells a complete thought (although it may make little sense taken in isolation out of context).

```
interface SentenceNode < Parent {
    type: "SentenceNode";
}
```

WordNode

Word (Parent) represents the smallest element that may be uttered in isolation with semantic or pragmatic content.

```
interface WordNode < Parent {
    type: "WordNode";
}
```

PunctuationNode

Punctuation (Parent) represents typographical devices which aid the understanding and correct reading of other grammatical units.

```
interface PunctuationNode < Parent {
    type: "PunctuationNode";
}
```

WhitespaceNode

White Space (Punctuation) represents typographical devices devoid of content, separating other grammatical units.

```
interface WhiteSpaceNode < PunctuationNode {
    type: "WhiteSpaceNode";
}
```

SOURCENODE

Source (Text) represents an external (non-grammatical) value embedded into a grammatical unit, for example a hyperlink or an emoticon.

```
interface SourceNode < Text {
    type: "SourceNode";
}
```

TEXTNODE

Text (Text) represents actual content in a NLAST document: One or more characters.

```
interface TextNode < Text {
    type: "TextNode";
}
```

B

PARSE-LATIN OUTPUT

An example of how parse-latin tokenises the paragraph “A simple sentence. Another sentence.”, is represented as follows,

```
{
  "type": "RootNode",
  "children": [
    {
      "type": "ParagraphNode",
      "children": [
        {
          "type": "SentenceNode",
          "children": [
            {
              "type": "WordNode",
              "children": [{
                "type": "TextNode",
                "value": "A"
              }]
            },
            {
              "type": "WhiteSpaceNode",
              "children": [{
                "type": "TextNode",
                "value": " "
              }]
            },
            {
              "type": "WordNode",
              "children": [{
                "type": "TextNode",
                "value": "simple"
              }]
            },
            {
              "type": "WhiteSpaceNode",
              "children": [{
                "type": "TextNode",
                "value": " "
              }]
            },
            {
              "type": "WordNode",
              "children": [{
                "type": "TextNode",
                "value": "sentence"
              }]
            },
            {
```

```
                "type": "PunctuationNode",
                "children": [{
                  "type": "TextNode",
                  "value": "."
                }]
              }
            ]
          },
          {
            "type": "WhiteSpaceNode",
            "children": [
              {
                "type": "TextNode",
                "value": " "
              }
            ]
          },
          {
            "type": "SentenceNode",
            "children": [
              {
                "type": "WordNode",
                "children": [{
                  "type": "TextNode",
                  "value": "Another"
                }]
              },
              {
                "type": "WhiteSpaceNode",
                "children": [{
                  "type": "TextNode",
                  "value": " "
                }]
              },
              {
                "type": "WordNode",
                "children": [{
                  "type": "TextNode",
                  "value": "sentence"
                }]
              },
              {
                "type": "PunctuationNode",
                "children": [{
                  "type": "TextNode",
                  "value": "."
                }]
              }
            ]
          }
        ]
      }
    ]
}
```

C

DOM

The DOM specification defines a platform-neutral model for errors, events, and (for this paper, the primary feature) node trees. XML-based documents can be represented by the DOM.

Consider the following HTML document:

```
<!DOCTYPE html>
<html class=e>
    <head><title>Aliens?</title></head>
    <body>Why yes.</body>
</html>
```

Is represented by the DOM as follows:

```
|- Document
   |- Doctype: html
   |- Element: html class="e"
      |- Element: head
      |  |- Element: title
      |     |- Text: Aliens?
      |- Text: "\n "
      |- Element: body
        |- Text: "Why yes.\n"
```

The DOM interfaces of bygone times were widely considered horrible, but newer features seem to be gaining popularity in the web authoring community as broader implementation across user agents is reached.

GLOSSARY

AJAX	Asynchronous JavaScript and XML. 8
API	Application Programming Interface. 8, 14
AST	Abstract Syntax Tree. 18
CI	Continuous Integration. 14
CSS	Cascading Style Sheets. 11, 17
CST	Concrete Syntax Tree. 18
DOM	Document Object Model. 20, 33
ECMAScript	More commonly known as JavaScript (which is in fact a proprietary eponym), ECMAScript is a language widely used for client-side programming on the web. vii, xi, 11, 14, 17, 19, 20
HTML	Hypertext Markup Language. 11, 20
HTML5	Hypertext Markup Language, version 5. 8
HTTP	Hypertext Transfer Protocol. 8
IBM	International Business Machines Corporation, a U.S. multinational technology and consulting corporation. 3
JSCS	JavaScript Code Style Checker. 13
JSDoc	JavaScript Documentation, markup language using comments to annotate ECMAScript source code. 14
MIT	Michigan Institute of Technology. 12
MongoDB	Mongo Database. 11
MySQL	My Structured Query Language. 11
NLCST	Natural Language Concrete Syntax Tree. ix, 17–20, 24, 27
NLP	Natural Language Processing. ix, xi, xii, 3–8, 11, 12, 17, 23
npm	Package manager for, and included in, Node.js. 15

PHP	PHP: Hypertext Preprocessor. 11
POS	Part-of-Speech. 3–5, 7, 21
TextOM	Text Object Model. ix, 17, 19, 20, 24

WORKS CITED

Ahmed, S. [sarfraznawaz]. 'New #dailyjs #javascript post : Natural Language Parsing with Retext http://ift.tt/1qrUjGo'. Twitter, 31/7/2014. Web. 8th Aug. 2014.

Anantheswaran, K. [gotwarlost]. 'istanbul'. *gotwarlost/istanbul*. GitHub, 5/8/2014, version 0.3.0. Web. 9th Aug. 2014.

'Annotating JavaScript for the Closure Compiler'. *Google Developers: Closure Tools*. Google, 30/6/2014. Web. 8th Aug. 2014.

'API Design Principles'. *Qt Project: Qt Wiki*. Qt Project, Digia Plc, 7/8/2014. Web. 8th Aug. 2014.

Balbin, J. 'TextTeaser'. *TextTeaser: An Automatic Summarization Application and API*. TextTeaser, 2014. Web. 8th Aug. 2014.

———[Mojojolo]. 'textteaser'. *Mojojolo/textteaser*. GitHub, 25/6/2014. Web. 9th Aug. 2014.

Baldridge, J. 'OpenNLP'. *Apache OpenNLP: Welcome to Apache OpenNLP!* Apache, 2005. Web. 9th Aug. 2014.

Bendersky, E. 'Abstract vs. Concrete Syntax Trees'. *Eli Bendersky's Website*. 16/2/2009. Web. 8th Aug. 2014.

Bird, S., E. Klein and E. Loper. *Natural Language Processing with Python*. 1st ed. Sebastopol, California: O'Reilly Media, 7/2009. Print.

Bloch, J. 'How to design a good API and why it matters'. *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM. 2006. 506–507. Print.

Brants, T. and A. Franz. '{Web 1T 5-gram Version 1}' (2006). Print.

Bringhurst, R. *The Elements of Typographic Style*. 4.0. Point Roberts, WA, USA: Hartley & Marks Publishers, 15/1/2013. Print.

Brooks, J. [jbrooksuk]. 'node-summary'. *jbrooksuk/node-summary*. GitHub, 18/2/2014, version 1.0.0. Web. 9th Aug. 2014.

Burkhead, J. [jlburkhead]. 'Spelling fix in README'. *wooorm/retext: Pull Request #11*. GitHub, 4/8/2014. Web. 8th Aug. 2014.

'Code Climate'. *Code Climate: Hosted static analysis for Ruby and JavaScript source code*. Bluebox, web. 8th Aug. 2014.

Cooper, P. 'Issue 193: August 8, 2014'. *JavaScript Weekly: A Free, Weekly Email Newsletter*. 8/8/2014. Web. 12th Aug. 2014.

———.'Issue 47: August 7, 2014'.
Node Weekly: A Free, Weekly Email Newsletter. 7/8/2014.
Web. 12th Aug. 2014.
dailyjs [dailyjs]. 'Natural Language Parsing with Retext:
http://dailyjs.com/2014/07/31/retext'. Twitter, 31/7/2014.
Web. 8th Aug. 2014.
Dulin, M. [mdevils]. 'JSCS'. *mdevils/node-jscs*.
GitHub, 8/8/2014, version 1.5.9. Web. 9th Aug. 2014.
Francis, W. N. and H. Kučera. 'Brown corpus manual'.
Brown University Department of Linguistics (1979). Print.
Gonzaga dos Santos Filho, L. [lfilho]. 'Is this English only?'
wooorm/retext: Issue #14. GitHub, 8/8/2014. Web. 9th Aug. 2014.
Grigorik, I. [igrigorik].
'English (latin) language parser in JavaScript: http://bit.ly/V8cCq7 -
aka, English > AST > transform and ... > profit. fun stuff!'
Twitter, 8/8/2014. Web. 9th Aug. 2014.
gut4. 'Some demos not working in chrome >36'.
wooorm/retext: Issue #10. GitHub, 1/8/2014. Web. 8th Aug. 2014.
Haverbeke, M. [marijnh]. 'acorn'. *marijnh/acorn*.
GitHub, 31/7/2014, version 0.6.0. Web. 9th Aug. 2014.
Hidayat, A. [ariya]. 'esprima'. *ariya/esprima*.
GitHub, 30/7/2014, version 1.2.2. Web. 9th Aug. 2014.
Holowaychuk, T. [visionmedia]. 'mocha'. *visionmedia/mocha*.
GitHub, 6/8/2014, version 1.21.4. Web. 9th Aug. 2014.
Holowaychuk, T., S. Baumgartner, J. Ong, K. Mårtensson, M. Bu,
M. Thirouin, N. Gallagher, A. Sexton and (dead-horse) [reworkcss].
'css'. *reworkcss/css*. GitHub, 5/8/2014, version 2.1.0.
Web. 9th Aug. 2014.
———[reworkcss]. 'rework'. *reworkcss/rework*.
GitHub, 25/6/2014, version 1.0.0. Web. 9th Aug. 2014.
Hunzaker, N. [nhunzaker]. 'speakeasy'. *nhunzaker/speakeasy*.
GitHub, 10/4/2013, version 0.2.11. Web. 9th Aug. 2014.
Hutchins, J. 'The first public demonstration of machine translation:
the Georgetown-IBM system, 7th January 1954'. Expanded version
of the paper presented at the AMTA conference in September 2004.
Print.
Hyder, Z. 'Gmail: 9 Years and Counting'. *Official Gmail Blog*.
Google, 10/4/2013. Web. 8th Aug. 2014.
JavaScript Daily [JavaScriptDaily].
'Retext: Extensible System for Analysing and Manipulating Natural
Language - https://github.com/wooorm/retext'. Twitter, 10/8/2014.
Web. 12th Aug. 2014.
Liddy, E. D. 'Natural language processing'.
Encyclopedia of Library and Information Science. 2nd ed.
NY: Marcel Decker, Inc., 2001. Print.
Loadfive [loadfive]. 'Knwl.js'. *loadfive/knwl.js*.
GitHub, 4/8/2014, version 0.0.1. Web. 9th Aug. 2014.
MacIntyre, R. 'Treebank tokenisation'. *Treebank tokenisation*.
University of Pennsylvania, 1995. Web. 14th Aug. 2014.

Miede, A. ‘classicthesis: A Classic Thesis Style for LaTeX and LyX’. *Google Project Hosting*. 13/8/2012, version 4.1. Web. 13th Aug. 2014.
Misiti, J. [josephmisiti]. ‘Awesome Machine Learning’. GitHub, 3/8/2014. Web. 8th Aug. 2014.
Mitkov, R., C. Orasan and R. Evans. ‘The importance of annotated corpora for NLP: the cases of anaphora resolution and clause splitting’. *Proceedings of Corpora and NLP: Reflecting on Methodology Workshop*. Citeseer. Cargese, Corse, 7/1999. 60–69. Print.
Mohri, M., A. Rostamizadeh and A. Talwalkar. *Foundations of Machine Learning (Adaptive Computation and Machine Learning series)*. MIT press, 8/2012. Print.
New York Times [NYTimes]. ‘Emphasis’. *NYTimes/Emphasis*. GitHub, 5/4/2012. Web. 9th Aug. 2014.
Newspaper.io. ‘JavaScript Daily - Issue 2014-08-11’. *Newspaper.io: Keeping you posted*. 11/8/2014. Web. 12th Aug. 2014.
Nielsen, F. Å. ‘AFINN’ (3/2011). Print.
O’Neil, J. ‘Doing Things with Words, Part Two: Sentence Boundary Detection’. *Attivo*. Attivo, 29/10/2008. Web. 8th Aug. 2014.
Oswald, T. [therebelrobot]. ‘This. Just. Made. My. Brain. Explode. https://github.com/wooorm/retext’. Twitter, 1/8/2014. Web. 8th Aug. 2014.
‘Outlook Web Access - A Catalyst for Web Evolution’. *The Exchange Team Blog: Blogs*. Microsoft Corp., 21/6/2005. Web. 8th Aug. 2014.
‘Parser API’. *Mozilla: MDN*. Mozilla, 29/7/2014. Web. 8th Aug. 2014.
Polencic, D. [danielepolencic]. ‘Extensible system for analysing and manipulating natural language’. *Node*. Reddit, 29/7/2014. Web. 8th Aug. 2014.
Princeton University. ‘About WordNet’. *WordNet: About WordNet*. Princeton University, 2010. Web. 8th Aug. 2014.
rbakhshi. ‘direction demo does not behave correctly’. *wooorm/retext: Issue #15*. GitHub, 11/8/2014. Web. 12th Aug. 2014.
Rinaldi, B. [remotesynth]. ‘Retext is a JavaScript natural language parser useful for doing things like removing profanity, analyzing text, etc. http://bit.ly/1zCZ1GP’. Twitter, 1/8/2014. Web. 8th Aug. 2014.
Roth, K. [thinkroth]. ‘Sentimental’. *thinkroth/Sentimental*. GitHub, 12/4/2014, version 1.0.1. Web. 9th Aug. 2014.
Schlicht, R. ‘microtype: CTAN’. *Package microtype*. 23/5/2013, version 2.5a. Web. 13th Aug. 2014.
Sliwinski, A. [thisandagain]. ‘sentiment’. *thisandagain/sentiment*. GitHub, 7/6/2014, version 1.0.1. Web. 9th Aug. 2014.
Sorhus, S. [sindresorhus]. ‘Awesome Node.js’. GitHub, 7/8/2014. Web. 8th Aug. 2014.
Spector, P. ‘Welcome to Interrobang-Mks’. *Welcome to Interrobang-Mks*. American Translators Association, 1/1/1996. Web. 8th Aug. 2014.

'Stanbol'. *Apache Stanbol: Welcome to Apache Stanbol!* Apache, 2010-12-14. Web. 9th Aug. 2014.
'Stargazers'. *wooorm/retext*. GitHub, 8/8/2014. Web. 8th Aug. 2014.
'Summly'. *Summly: Pocket sized news for iPhone*. Summly, 2012. Web. 8th Aug. 2014.
Suzuki, Y. [constellation]. 'escodegen'. *constellation/escodegen*. GitHub, 28/7/2014, version 1.3.3. Web. 9th Aug. 2014.
'TextRazor'. *TextRazor: The Natural Language Processing API*. TextRazor, 2014. Web. 8th Aug. 2014.
'Travis'. *Travis*. Travis CI GmbH, 2014. Web. 8th Aug. 2014.

Umbel, C., R. Ellis and K. Koch. 'natural'. *NaturalNode/natural*. GitHub, 22/6/2014, version 0.1.28. Web. 9th Aug. 2014.
Wormer, T. [wooorm_]. 'DailyJS: Natural Language Parsing with Retext'. *JavaScript*. Reddit, 5/8/2014. Web. 8th Aug. 2014.
———[wooorm_]. 'Natural Language Parsing with Retext'. *Node*. Reddit, 1/8/2014. Web. 8th Aug. 2014.
———[wooorm]. 'retext-directionality'. *wooorm/retext-directionality*. GitHub, 22/7/2014, version 0.1.1. Web. 9th Aug. 2014.
———[wooorm]. 'retext-double-metaphone'. *wooorm/retext-double-metaphone*. GitHub, 13/7/2014, version 0.1.0. Web. 9th Aug. 2014.
———[wooorm]. 'retext-emoji'. *wooorm/retext-emoji*. GitHub, 13/7/2014, version 0.1.0. Web. 9th Aug. 2014.
———[wooorm]. 'retext-keywords'. *wooorm/retext-keywords*. GitHub, 16/7/2014, version 0.0.1. Web. 9th Aug. 2014.
———[wooorm]. 'retext-porter-stemmer'. *wooorm/retext-porter-stemmer*. GitHub, 13/7/2014, version 0.1.0. Web. 9th Aug. 2014.
———[wooorm]. 'retext-pos'. *wooorm/retext-pos*. GitHub, 31/7/2014, version 0.1.2. Web. 9th Aug. 2014.
———[wooorm]. 'retext-search'. *wooorm/retext-search*. GitHub, 13/7/2014, version 0.1.0. Web. 9th Aug. 2014.
———[wooorm]. 'retext-smartypants'. *wooorm/retext-smartypants*. GitHub, 13/7/2014, version 0.1.0. Web. 9th Aug. 2014.
———[wooorm]. 'retext-visit'. *wooorm/retext-visit*. GitHub, 7/7/2014, version 0.1.0. Web. 9th Aug. 2014.
Young, A. 'Natural Language Parsing with Retext'. *DailyJS: A JavaScript Blog*. 31/7/2014. Web. 8th Aug. 2014.
Zakas, N. [eslint]. 'eslint'. *eslint/eslint*. GitHub, 6/8/2014, version 0.7.4. Web. 9th Aug. 2014.
Zimmerman, M. [mileszim]. 'sediment'. *mileszim/sediment*. GitHub, 9/9/2013, version 0.9.2. Web. 9th Aug. 2014.