breaklinkstrue

TITUS E. C. WORMER

RETEXT

RETEXT

TITUS E. C. WORMER

Design of an extensible system for analysing and manipulating natural language

August 2014 – version 0.3

Titus E. C. Wormer: *Retext*, Design of an extensible system for analysing and manipulating natural language, © August 2014

ACKNOWLEDGMENTS

Thanks to my supervisor Justus for the trust in me and my work, and for allowing me to produce a product and thesis that perhaps not every CMD professional understands, but most certainly falls within our field.

Some cool dedication… Thanks all!

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.

INTRODUCTION

Natural Language Processing (NLP), a field of computer science, artificial intelligence, and linguistics concerned with the interactions between computers and human languages (see 'Natural language processing'), is becoming increasingly important in society. For example, search engines try to answer before a question is given, the NSA detects terrorist-toned motifs in text messages, and e-mail applications know if a user forgot to attach an attachment.

However, to date, web developers trying to solve NLP problems reinvent the wheel over and over again. There are certainly tools (especially for other platforms. such as Python (see Bird, Klein and Loper) and Java (see Baldridge)) trying to solve this, 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 for multipurpose natural language analysis—a standard representation of the grammatical hierarchy of text.

For over a year I too have camped with the previously mentioned problem. I have tried to solve it, to no avail, on numerous occasions. During the creation of this thesis I developed a well thought out and substantiated solution, which solves the previously mentioned problems.

The in this thesis introduced implementation Retext (and other projects in the Retext family) are a new approach to the syntax of natural text. My goal in this paper is to design an extensible system for multipurpose analysis of language.

To achieve this goal, I have organized my paper into X main sections
In the first section … In the second section, …
I end my paper with a third section that offers …
…and conclude with a fourth section that …
I also include an appendix after the References that contains …
Before I can begin the examination of …, however, I need to provide … {next section}

1 For example, ignoring white space (See Loadfive), punctuation, or implementing a naive definition of "words" (such as Hunzaker), or by deploying a less than adequate algorithm to detect sentences (such as 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, Ellis and Koch), where modularity and frugality are in order.

CONTENTS

i RETEXT 1

1 CONTEXT 3
- 1.1 Scope . . . 4
- 1.2 Implementations . . . 4
  - 1.2.1 Stages . . . 5
  - 1.2.2 Applications . . . 6
  - 1.2.3 Using Corpora for NLP . . . 7
  - 1.2.4 Using a web API . . . 8

2 DESIGN 11
- 2.1 Architecture . . . 11
  - 2.1.1 Syntax: NLCST . . . 11
  - 2.1.2 Parser: Parse-latin . . . 12
  - 2.1.3 Object Model: TextOM . . . 13
  - 2.1.4 Natural Language System: Retext . . . 14

3 PRODUCTION 15
- 3.1 Target Audience . . . 15
- 3.2 Use cases . . . 15
- 3.3 Requirements . . . 16
  - 3.3.1 Open Source . . . 16
  - 3.3.2 Performance . . . 16
  - 3.3.3 Testing . . . 17
  - 3.3.4 Code quality . . . 17
  - 3.3.5 Automation . . . 18
  - 3.3.6 API Design . . . 18
  - 3.3.7 Installation . . . 19

4 VALIDATION 21
- 4.1 Plugins . . . 21
- 4.2 Reception . . . 21

5 CONCLUSION 23
- 5.1 Summary . . . 23
- 5.2 Future Work . . . 23
- 5.3 Advice . . . 23

ii APPENDIX 25

A APPENDIX A: NLCST DEFINITION 27

B APPENDIX B: PARSE-LATIN OUTPUT 31

C APPENDIX C: DOM 35

GLOSSARY 37

WORKS CITED 39

Part I

RETEXT

1

CONTEXT

{{Needs a nice quote about the definition of NLP}}

The focus of this paper is Natural Language Processing (NLP). NLP concerns itself with enabling machines to understand human language, thus it is a field related to human–computer interaction. Human language, a medium which humans understand quite easily, can pose problems for machines.

That understanding of human language by machines is quite difficult, is depicted by one of the first application of NLP, during the Georgetown–IBM experiment, where a Russian-english machine translation system was showcased in New York in 1945 (Hutchins). More than sixty sentences were translated by a machine from Russian to English. The experiment was well publicised in the press and resulted in optimism for machine-translation. Machine-translation was thought of as being a solved problem within three to five years. The results in the following ten years were however disappointing and eventually led to reduced funding.

Machine translation is just one of many major tasks involved with NLP. Other applications include automatic summarisation (generating a summary), named entity recognition (detecting references to people or places), sentiment analysis (extracting opinion).

Summarisation, named entity recognition, and sentiment analyses are all part of what is known as "information extraction", a term for the act of finding certain data inside a natural language document. These, and many other applications of NLP, are implemented widely by many programs. The approach taken for such a goal is mostly equivalent. For example, the information extraction method for entity linking in a text document could be as follows (according to 'Stanbol'):

- Detect the input language — *Language Detection*: Based on the language of the given text, the algorithms behind the following steps will change significantly. Often omitted if only a single language is supported.
- Optionally break sentences — *Sentence Tokenisation*: Breaking sentences can elevate performance and heighten the accuracy of the following stages (in particular Part-of-Speech (POS) tagging);

- Breaking words — *Word Tokenisation*: To detect entities, one must of course break those entities (words) free from their surroundings;
- Detecting word-categories — *Part-of-Speech (POS) Tagging*: When linking entities in an encyclopaedia, one would typically not want to link every entity to another entry, but most often only nouns or even proper nouns. POS tagging helps make that decision;
- Optionally detecting noun phrases — *Noun Phrase Detection*: Although "apple" and "juice" are both words and could both be linked to separate entities, it would be more apt to link both words to one entity: "apple juice", noun phrase detection makes this possible;
- Optionally detecting lemmas or stems — *Lemmatisation or Stemming*: All forms of walk could link to the same entity, be it "walk", "walked", or "walking". Detecting the lemma or stem for a word makes this possible;
- Linking the entities to their reference — *Entity Linking*.

Although many different fields are covered by NLP, the process of reaching those goals touches on some well defined stages as seen above.

1.1 SCOPE

The stages mentioned in the previous section are implemented in many NLP applications. This paper (the captured use cases and requirements, the proposed standard, and the example implementation) however will only cover one stage: tokenisation. Tokenisation here includes sentence boundary detection, word tokenisation, and tokenisation of other grammatical units.

Another decision made in scoping the NLP problem is that that this paper lays its focus on syntactic, and largely ignores semantic units (i.e., phrases and clauses).

Lastly, this paper ignores spoken language, and lays its focus on Latin script language: written languages using an alphabet depending on the letters of the classical Latin alphabet.

1.2 IMPLEMENTATIONS

While researching algorithms to tokenise natural language few viable options were found. Most algorithms look at one or two (or both) kinds of tokenisation: sentence tokenisation and word tokenisation. In this section the current implementations, where they excel, and what they lack.

1.2.1 *Stages*

1.2.1.1 *Sentence tokenisation*

Often referred to as sentence boundary disambiguation[1], sentence tokenisation is a very basic, but important part of NLP. It is almost always a stage in an NLP application and not an end goal. Sentence tokenisation helps subsequent stages (e.g., detecting plagiarism or POS tagging) work more accurately.

Oftentimes, one of three symbols is used to denote the end of a sentences: Either a full stop (.), an interrogative point (?), or an exclamation point (!)[2]. Detecting sentence boundaries is however not so easy as simply breaking sentences at these markers. The previously mentioned terminal markers might have other purposes: full stops are often suffixed to abbreviations or titles, in numbers, included in initialisms[3], or in embedded content[4]. The interrogative- and exclamation points too can occur ambiguously, most often in a quote (e.g., ' "Of course!", she screamed'). Disambiguation gets even harder when the previous mentioned exceptions are in fact also a sentence boundary, such as in "…use the feminine form of idem, ead." or in ' "Of course!", she screamed, "I'll do it!" ', where in both examples the last terminal markers end their respective sentence.

1.2.1.2 *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.

Frequently, words are seen as everything that is not white space (i.e., spaces, tabs, feeds), thereby their boundaries denoted by those white spaces[5]. A smarter algorithm would also treat punctuation marks as word boundaries, but such a general rule would wrongly classify inter-

1 Both sentence tokenisation and sentence boundary solve a similar issue—detecting where sentences terminate. Sentence boundary disambiguation focusses on the position where a sentence breaks (as in, 'One sentence.| Two sentences.|', where the pipe symbol refers to the end of one sentence and the beginning of another), whereas sentence tokenisation lays focus on where a sentence starts and where it ends (as in, '{One sentence.} {Two sentences.}', where everything between curly brackets, both start and end boundaries, are detected and classified as s sentence).

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

3 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.").

4 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.

5 For example, Knwl counts words using this method (Loadfive).

word punctuation as not being part of words[6]. Many exceptions to this rule exist, e.g., hyphenation points, colons, or elision (whether full stops, "e.g."; apostrophes, the Dutch "'s"; or slashes, "N/A").

1.2.2 *Applications*

In the previous section implementations were covered that solve tokenisation stages in NLP applications, such as Natural's word tokenisers (Umbel, Ellis and Koch). Concluded was that these implementations are lacking. In this section several implementations are covered that solve these tokenisation stages as part of a larger natural language application.

1.2.2.1 *Sentiment Analysis*

Sentiment analysis is an NLP task which is concerned with the polarity (positive, negative) and subjectivity (objective, subjective) of text. Its application could look as follows.

1. Detect language — Often omitted if only a single language is supported.
2. Optionally tokenise sentences — Different sentences could have different sentiments, tokenising them helps provide better outcomes.
3. Tokenise words — Needed to compare with the database (see the last stage).
4. Optionally detect lemmas or stems — Might help classification.
5. Detect sentiment — Typically, sentiment analysers include a database mapping words, stems, or lemmas to their respective polarity and/or subjectivity[7].

Many implementations exist for this task (e.g., Roth; Zimmerman; Sliwinski), many of which exclude non-alphabetic characters form their definition of words, resulting in less than perfect results[8].

1.2.2.2 *Automatic Summarisation*

Automatic summarisation is an NLP task concerned with reducing a text in order to create a summary of the major points retaining the original document. In contrast with sentiment analysers, implementations of automatic summarisation algorithms on the web are less ubiquitously available[9].

6 For example, Natural's tokenisers (Umbel, Ellis and Koch).

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

8 If fact, all of the researched implementations deploy lacking tokenisations steps. Controversial, as they each create unreachable code through their naivety: all tested libraries remove dashes from words, while words such as "self-deluded" are in the databases they use, but never reachable.

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

An example application for automatic summarisation could be as follows.

1. Detect language — Often omitted if only a single language is supported.
2. Tokenise sentences — 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. Tokenise words — Used to calculate keywords (words which occur more often than expected by chance alone).
4. Automatic summarisation — Ranking the tokenised grammatical units (sentences, phrases) and return the highest ranking units.

Several factors can be used to determine whether or not a phrase or sentence should be included from a summary:

1. Number of words (an ideal sentence is neither too long nor too short);
2. Number of keywords (words which occur more often than expected by chance alone in the whole text);
3. Number of words from the document's title it contains;
4. Position inside a paragraph (initial and final sentences are often more important than sentences buried somewhere in the middle of a paragraph).

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

The only implementation working on the web (Brooks), takes a naive approach to sentence tokenisation. For example, ignoring sentences terminated by an exclamation point. Note that both other implementations, and many more, use a whole different approach to sentence tokenisation: Corpora.

1.2.3 *Using Corpora for NLP*

A corpus is a large, structured set of texts used for many linguistic and NLP tasks. Oftentimes, each item (most frequently words, but some corpora include sentences, phrases, clauses, or other units) in a corpus is annotated ("tagged") with information about the item (for example, part-of-speech tags or lemmas).

These huge (often more than a million words[10]) chunks of information are the basis of many of the newer revolutions in NLP (see Mitkov, Orasan and Evans). Supervised learned[11] parsing (in NLP, based on an-

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

11 "[From] a set of labeled examples as training data … [, make] predictions for all unseen points" (Mohri, Rostamizadeh and Talwalkar).

notated corpora), is the direct antagonist of rule-based parsing[12]. Instead of defining rules, exceptions to these rules, exceptions to these exceptions, and so on, supervised learning delegates this task to machines, creating a more performant, scalable, program.

Generally, corpus linguistics have proven to be better in several ways over rule-based linguistics. However, they do have their disadvantages:

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

Most important is the fact that supervised learning will not fare well on the web: Loading corpora over the network each time a user request a web page is unfeasible for most web sites or applications[13].

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

1.2.4 *Using a web API*

Where the term Application Programming Interface (API) stands for the interface between two programs, the term is often defined in the context of 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 "Tweets" and other objects (users, images, &c.). In the context of this paper, the term Web API is used for the latter, whereas API is used for any programming interface.

With the rise of the asynchronous web[14], supervised learning too became widely available through web APIs (e.g., Balbin, 'TextTeaser'; Princeton University; 'TextRazor'). The new web APIs made it possible to implement supervised learning techniques on the web, without the need for downloading a corpus to your or your users' computer.

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

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

13 Currently, almost no technologies exist for storing large datasets in a browser. The only exception to the rule is the HTML5 File System API, but development has stopped (U).

14 Starting around 2000 Asynchronous JavaScript and XML (AJAX) started to transform the web. Beforehand, websites only really changed 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".

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

2

DESIGN

2.1 ARCHITECTURE

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

- Natural Language Concrete Syntax Tree (NLCST) — Defines a standard for classifying grammatical units understandable for machines;
- Parse-latin — Classifies natural language according to NLCST;
- Text Object Model (TEXTOM) — Provides an interface for analysing and manipulating output provided by parse-latin;
- 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 provide their own software to replace one of the subproposals. For example, other parties can create a parser for the Chinese language and use it instead of parse-latin to classify natural language according to NLCST.

2.1.1 *Syntax: NLCST*

To develop natural language tools in ECMAScript, an intermediate representation of natural language is useful: instead of each module defining their own representation of text, using a single syntax leads to better results, interoperability, and performance.

The elements defined by NLCST (Natural Language Concrete Syntax Tree) are based on the the grammatical hierarchy, but by default do not expose all its constituents[1]. Additionally, more elements are provided to cover other semantic units in natural language[2].

The definitions of exposed nodes were heavily based on other specifications of syntax trees for manipulation on the web platform, such

1 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.

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

as CSS, aptly named for the CSS language (Holowaychuk et al., 'css') or the Mozilla JavaScript AST, for ECMAScript ('Parser API'). Both implementations are widely used. CSS by Rework (Holowaychuk et al., 'rework'), and Mozilla JavaScript Abstract Syntax Tree (AST) by Esprima (Hidayat), Acorn (Haverbeke), and Escodegen (Suzuki).

Note that the aforementioned syntax tree specifications are both ASTS, whereas NLCST is a a Concrete Syntax Tree (CST). A concrete syntax tree is a one-to-one mapping of source to result. All information stored in the original input is also available through the resulting tree (Bendersky).

The information stored in CSTS is very verbose and could lead to trees that are hard to work with. On the other hand, the fact that every part of the input is housed in the tree, makes it easy for developers to save the output or pass it on to other libraries for further processing.

See the appendices for a complete list of specified nodes of NLCST.

2.1.2 *Parser: Parse-latin*

ECMAScript is used extensively. Because of this, many ECMAScript tools are being developed. This includes tools for Natural Language Processing. These ECMAScript tools however, when run on the client-side, can not implement supervised learning based on corpora, and web API usage too is not ideal. Thus, a rule-based parser is needed to tokenise text.

For creating such intermediate representations from Latin-script based languages, parse-latin is presented in this paper[3]. As proof-of-concept, two other libraries are also presented, parse-english and parse-dutch, using parse-latin as a base and providing better support for several language specific features, respectively English and Dutch.

By following NLCSTS, modules building on parse-latin may receive better results or performance over implementing their own parsing tools.

By using the CST as described by NLCST and the parser as described by parse-latin, the intermediate representation can be used by developers to create independent modules.

Basically, parse-latin splits text into white space, word, and punctuation tokens. parse-latin starts out with a pretty simple definition, one that most other tokenisers use:

- A "word" is one or more letter or number characters;
- A "white space" is one or more white space characters;
- A "punctuation" is one or more of anything else;

Then, it manipulates and merges those tokens into a syntax tree, adding sentences and paragraphs where needed.

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

- Some punctuation marks are part of the word they occur in, e.g., “non-profit”, “she’s”, “G.I.”, “11:00”, “N/A”;
- Some full-stops do not mark a sentence end, e.g., “1.”, “e.g.”, “id.”;
- Although full-stops, question marks, and exclamation marks (sometimes) end a sentence, that end might not occur directly after the mark, e.g., “.)”, ’.“’;

See the appendices for example output provided by the parse-latin parser.

2.1.2.1 *parse-english*

parse-english has the same interface as parse-latin, but returns results better suited for English natural language. For example:

- Unit abbreviations (“tsp.”, “tbsp.”, “oz.”, “ft.”, &c.);
- Time references (“sec.”, “min.”, “tues.”, “thu.”, “feb.”, &c.);
- Business Abbreviations (“Inc.” and “Ltd.”);
- Social titles (“Mr.”, “Mmes.”, “Sr.”, &c.);
- Rank and academic titles (“Dr.”, “Rep.”, “Gen.”, “Prof.”, “Pres.”, &c.);
- Geographical abbreviations (“Ave.”, “Blvd.”, “Ft.”, “Hwy.”, &c.);
- American state abbreviations (“Ala.”, “Minn.”, “La.”, “Tex.”, &c.);
- Canadian province abbreviations (“Alta.”, “Qué.”, “Yuk.”, &c.);
- English county abbreviations (“Beds.”, “Leics.”, “Shrops.”, &c.);
- Common elision (omission of letters) (“ ‘n’ ”, “’o”, “’em”, “’twas”, “’80s”, &c.).

2.1.2.2 *parse-dutch*

parse-dutch has, just like parse-english, the same interface as parse-latin, but returns results better suited for Dutch natural language. For example:

- Unit and time abbreviations (“gr.”, “sec.”, “min.”, “ma.”, “vr.”, “vrij.”, “febr”, “mrt”, &c.);
- Many other common abbreviations: (“Mr.”, “Mv.”, “Sr.”, “Em.”, “bijv.”, “zgn.”, “amb.”, &c.);
- Common elision (omission of letters) (“d”’, “’n”, “’ns”, “’t”, “’s”, “’er”, “’em”, “’ie”, &c.).

2.1.3 *Object Model: TextOM*

To modify NLCST nodes in ECMAScript, this paper proposes TEXTOM. TEXTOM implements the nodes defined by NLCST, but provides an object-oriented style[4]. TEXTOM was designed to be similar to the Document

4 Object-oriented programming is a style of programming, where classes, instances, attributes, and methods are important.

Object Model (DOM)[5], the mechanism used by browsers to expose Hypertext Markup Language (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 events (a mechanism for detecting changes), modification functionality (inserting, removing, and replacing children into/from parents), and traversal (e.g., finding all words in a sentence).

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

2.1.4 *Natural Language System: Retext*

For natural language processing on the client side, this paper proposes Retext. Retext combines a parser, such as parse-latin or parse-dutch, with a manipulatable object model (TextOM).

In addition, Retext provides a minimalistic plugin mechanism so developers can create and publish plugins for others, and in turn can use others' plugins inside their projects.

5 See the appendices for a more information on the DOM.

3

PRODUCTION

3.1 TARGET AUDIENCE

The audience that would benefit the most from such a solution are web developers: programmers who specialise in creating software for the world wide web. Web developers 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 HTML, graphic design through Cascading Style Sheets (CSS), creating a back end in node.js, PHP: Hypertext Preprocessor (PHP), &c., contacting a database through Mongo Database (MONGODB), My Structured Query Language (MYSQL), and more. In addition, many interdisciplinary skills are also of concern to web developers, such as usability, accessibility, copywriting, information architecture, optimisation.

3.2 USE CASES

The use cases of the target audience, the front-end 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 can be categorised into three broad fields: analysation, manipulation, and creation.

- The practitioner may intent summarise natural text (mostly analysation, potentially also manipulation);
- The practitioner 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);
- The practitioner may intent to recognise sentiment in text: is a Tweet positive, negative, or spam? (analysation);
- The practitioner may intent to replace so-called "dumb" punctuation with so-called "smart" punctuation, e.g., dumb quotation with (") or ("); three dots with an ellipsis (…), or two hyphens with an en-dash (–) (manipulation);

- The practitioner may intent to count the number of certain elements in the grammatical hierarchy in a document, e.g, words, white space, punctuation, sentences, or paragraphs (analysation);
- The practitioner may intent to recognise which language a given document is written in (analysation);
- The practitioner may intent to find words 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).

Natural Language Processing is a broad field concerned with the interactions between computers and human languages, with bases in computer science, artificial intelligence, and linguistics. NLP poses may challenges, but not every challenge challenge in the field is of interest to the web developer—the developer enabling machines to respond to humans through the world wide web. Most importantly, the more academic areas of NLP do not fit well with the goals of web developers, e.g., speech recognition, optical character recognition, text-to-speech transformation, translation, or machine learning.

3.3 REQUIREMENTS

The in the previous section covered use cases are required to be reachable through the proposed solution. In addition, this section covers other requirements to better suit the wishes of the target audience.

3.3.1 *Open Source*

To reach the target audience and validate its use, the software was developed as open source software—software that is free for all to use and redistribute. All projects were licensed under The MIT License ('The MIT License (MIT)'), a license which provides rights for others to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of these projects.

In addition, the software was developed under the all-seeing-eye of the GitHub community. GitHub is a hosted version control[1] service with social networking features. On GitHub, many web developers follow their peers to track what they are working on, watch their favourite projects to get notified of changes, and raise issues or feature requests.

3.3.2 *Performance*

The in this paper proposed implementations were produced with high regards for performance. Performance includes the software having a

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

small file size in order 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.

3.3.3 *Testing*

With the development of the in this paper introduced software, testing was given high priority. Testing, in software development, refers to validating if software does what it is supposed to do, and can be divided into multiple subgroups:

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

Great care was given to develop a full 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 were run through Mocha (Holowaychuk), coverage was detected with Istanbul (Anantheswaran).

3.3.4 *Code quality*

Great attention was given to code quality: how useful and readable for both humans and machines the software is. Special focus was given to consistency and clearness for humans.

3.3.4.1 *Suspicious Code and Bugs*

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

3.3.4.2 *Style*

To enforce a consistent code style, in order to create software readable for humans, JavaScript Code Style Checker (JSCS) was used (Dulin). JSCS provides rules for allowing or disallowing patters such as white space at the end of a line or camel cased variable names, or setting a maximum line length. JSCS was chosen because it, just like the aforementioned Eslint, provides a strong base set of rules. The rules chosen for the development of the proposed software was set very strict to enforce all code was written in the same manner.

3.3.4.3 *Commenting*

Even when code is completely bug free, uses no hard-to-understand shortcuts, and adheres to a strict style, it might still be hard to understand for humans. The act of commenting code—describing what a program does and why it accomplishes this in a certain way—is important. On the other hand, commenting can also be to verbose, for example when the code is duplicated in natural language.

JavaScript Documentation (JSDOC) ('Annotating JavaScript for the Closure Compiler') is a markup language for ECMAScript programs, allowing developers to embed documentation in source code. Later various tools can be used to extract the documentation and expose it separately from the original code.

Great care was given to annotate "tricky" source code inside the software with comments, and to apply documentation inside the source code through JSDOC.

3.3.5 *Automation*

Great focus was given to develop using several automated Continuous Integration (CI) environments. When suspicious, ambiguous, or buggy code was introduced in the software, the error was automatically detected and in some cases deployment was prevented.

Tools user were Code Climate ('Code Climate') to detect complex, duplicate, or bug-prone code, and Travis ('Travis') to validate all unit tests passed before deploying the software.

3.3.6 *API Design*

Interface design was given high priority for the development of the proposed software. A clear interface of the software, according to Joshua Bloch ('How to design a good API and why it matters'), has the following characteristics:

- Easy to learn;
- Easy to use;
- Hard to misuse;
- Easy to read;
- Easy to maintain;
- Easy to extend;
- Meeting its requirements;
- 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'), are as follows:

- Be minimal;

- Be complete;
- Have clear and simple semantics;
- Be intuitive;
- Be easy to memorise;
- Lead to readable code;

With the creation of the software these characteristics, and the in their sources given examples, were taken into account.

3.3.7 *Installation*

Access both on the client side and on the server side to the software was of importance during the development of the software. For the server side on Node.js, Node Package Manager (NPM)—the default package manager for the platform—is the most popular. On the client side, many different package managers exist, the most popular[2] being Bower and Component. To reach the target audience, in addition of making the whole source available for download through Git and GitHub, NPM, Bower, and Component were used.

2 Popularity here is simply defined as having the most search results on Google.

4

VALIDATION

The developed software was validated through two approaches. The design and usability of the interface was validated by trying to solve the use cases with the software. If the target audience actually wanted to use the software was validated through enthusiasm showed by the open source community.

4.1 PLUGINS

More than fifteen (15) Retext plugins were created to validate how the different projects integrated together, and how the system worked as a whole.

The created plugins include tools for:

- Transforming so-called dumb punctuation marks into more typographically correct punctuation marks;
- Transforming emoji short-codes (:cat:) into real emoji;
- Detecting the direction of text (left to right, right to left);
- Detecting phonetics (how words sound like);
- Detecting the stem of words (reducing inflected or derived words to a single form);
- Detecting grammatical units (to count words in a sentence);
- Finding text (even misspelled) in a document;
- Detecting part-of-speech tags of words;
- Finding keywords and key phrases in a document;

The creation of these plugins brought several problems to light in the developed software. These problems were then dealt with back and forth between the software and the plugins. The software changed severely by these fixed problems, which resulted in a more useable interface.

4.2 RECEPTION

To validate if the target audience actually wanted to use the developed software, 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 several mentions on blogs[1], link roundups[2], reddit[3]. In turn, these publications resulted in positive reactions (such as on Twitter[4]), feedback (gut4), and fixes (Burkhead) on these mentions, and many project-followers on GitHub ('Stargazers').

1 (Young)
2 Including (Misiti) and (Sorhus).
3 Including (Polencic), (Wormer, 'Natural Language Parsing with Retext'), and ('DailyJS: Natural Language Parsing with Retext').
4 Including (dailyjs), (Ahmed), (Oswald), (Rinaldi), and more.

5

CONCLUSION

5.1 SUMMARY

5.2 FUTURE WORK

5.3 ADVICE

Part II

APPENDIX

A

APPENDIX A: NLCST DEFINITION

NODE

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

PARENT

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

TEXT

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

LOCATION

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

POSITION

```
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

APPENDIX 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

APPENDIX 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

ACRONYMS

AJAX	Asynchronous JavaScript and XML. 9
API	Application Programming Interface. 8, 9, 12, 18
AST	Abstract Syntax Tree. 12
CI	Continuous Integration. 18
CSS	Cascading Style Sheets. 12, 15
CST	Concrete Syntax Tree. 12
DOM	Document Object Model. 13, 14, 35
HTML	Hypertext Markup Language. 14, 15
HTML5	Hypertext Markup Language 5. 8
HTTP	Hypertext Transfer Protocol. 8
IBM	International Business Machines Corporation. 3
JSCS	JavaScript Code Style Checker. 17
JSDOC	JavaScript Documentation. 18
MIT	Michigan Institute of Technology. 16
MONGODB	Mongo Database. 15
MYSQL	My Structured Query Language. 15
NLCST	Natural Language Concrete Syntax Tree. 11–14
NLP	Natural Language Processing. xi, 3–9, 11, 15, 16
NPM	Node Package Manager. 19
PHP	PHP: Hypertext Preprocessor. 15
POS	Part-of-Speech. 3–5

TextOM Text Object Model. 11, 13, 14

REFERENCES

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

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

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

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

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

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

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

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

Bird, Steven, Ewan Klein and Edward Loper. *Natural Language Processing with Python*. 1st ed. Sebastopol, California: O'Reilly Media, July 2009. Print.

Bloch, Joshua. '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, Thorsten and Alex Franz. '{Web 1T 5-gram Version 1}' (2006). Print.

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

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

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

dailyjs [dailyjs]. 'Natural Language Parsing with Retext: http://dailyjs.com/2014/07/31/retext'. Twitter, 31st July 2014. Web. 8th Aug. 2014.

Dulin, Marat [mdevils]. 'JSCS'. *mdevils/node-jscs*. GitHub, 8th Aug. 2014, version 1.5.9. Web. 9th Aug. 2014.

Francis, W. Nelson and Henry Kučera. 'Brown corpus manual'. *Brown University Department of Linguistics* (1979). Print.

gut4. 'Some demos not working in chrome >36'. *wooorm/retext: Issue #10*. GitHub, 1st Aug. 2014. Web. 8th Aug. 2014.

Haverbeke, Marijn [marijnh]. 'acorn'. *marijnh/acorn*. GitHub, 31st July 2014, version 0.6.0. Web. 9th Aug. 2014.

Hidayat, Ariya [ariya]. 'esprima'. *ariya/esprima*. GitHub, 30th July 2014, version 1.2.2. Web. 9th Aug. 2014.

Holowaychuk, TJ [visionmedia]. 'mocha'. *visionmedia/mocha*. GitHub, 6th Aug. 2014, version 1.21.4. Web. 9th Aug. 2014.

Holowaychuk, TJ, et al. [reworkcss]. 'css'. *reworkcss/css*. GitHub, 5th Aug. 2014, version 2.1.0. Web. 9th Aug. 2014.

———[reworkcss]. 'rework'. *reworkcss/rework*. GitHub, 25th June 2014, version 1.0.0. Web. 9th Aug. 2014.

Hunzaker, Nathan [nhunzaker]. 'speakeasy'. *nhunzaker/speakeasy*. GitHub, 10th Apr. 2013, version 0.2.11. Web. 9th Aug. 2014.

Hutchins, John. '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, Zohair. 'Gmail: 9 Years and Counting'. *Official Gmail Blog*. Google, 10th Apr. 2013. Web. 8th Aug. 2014.

Loadfive [loadfive]. 'Knwl.js'. *loadfive/knwl.js*. GitHub, 4th Aug. 2014, version 0.0.1. Web. 9th Aug. 2014.

Misiti, Joseph [josephmisiti]. 'Awesome Machine Learning'. GitHub, 3rd Aug. 2014. Web. 8th Aug. 2014.

Mitkov, Ruslan, Constantin Orasan and Richard 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, July 1999. 60–69. Print.

Mohri, Mehryar, Afshin Rostamizadeh and Ameet Talwalkar. *Foundations of Machine Learning (Adaptive Computation and Machine Learning series)*. MIT press, Aug. 2012. Print.

'Natural language processing'. *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc., 5th Aug. 2014. Web. 8th Aug. 2014.

New York Times [NYTimes]. 'Emphasis'. *NYTimes/Emphasis*. GitHub, 5th Apr. 2012. Web. 9th Aug. 2014.

Nielsen, Finn Årup. 'AFINN' (Mar. 2011). Print.

O'Neil, John. 'Doing Things with Words, Part Two: Sentence Boundary Detection'. *Attivo*. Attivo, 29th Oct. 2008. Web. 8th Aug. 2014.

Oswald, Trent [therebelrobot]. 'This. Just. Made. My. Brain. Explode. https://github.com/wooorm/retext'. Twitter, 1st Aug. 2014. Web. 8th Aug. 2014.

'Outlook Web Access - A Catalyst for Web Evolution'. *The Exchange Team Blog: Blogs*. Microsoft Corp., 21st June 2005. Web. 8th Aug. 2014.

'Parser API'. *Mozilla: MDN*. Mozilla, 29th July 2014. Web. 8th Aug. 2014.

Polencic, Daniele [danielepolencic]. 'Extensible system for analysing and manipulating natural language'. *Node*. Reddit, 29th July 2014. Web. 8th Aug. 2014.

Princeton University. 'About WordNet'. *WordNet: About WordNet*. Princeton University, 2010. Web. 8th Aug. 2014.

Rinaldi, Brian [remotesynth]. 'Retext is a JavaScript natural language parser useful for doing things like removing profanity, analyzing text, etc. http://bit.ly/1zCZ1GP'. Twitter, 1st Aug. 2014. Web. 8th Aug. 2014.

Roth, Kevin [thinkroth]. 'Sentimental'. *thinkroth/Sentimental*. GitHub, 12th Apr. 2014, version 1.0.1. Web. 9th Aug. 2014.

Sliwinski, Andrew [thisandagain]. 'sentiment'. *thisandagain/sentiment*. GitHub, 7th June 2014, version 1.0.1. Web. 9th Aug. 2014.

Sorhus, Sindre [sindresorhus]. 'Awesome Node.js'. GitHub, 7th Aug. 2014. Web. 8th Aug. 2014.

Spector, Penny. 'Welcome to Interrobang-Mks'. *Welcome to Interrobang-Mks*. American Translators Association, 1st Jan. 1996. Web. 8th Aug. 2014.

'Stanbol'. *Apache Stanbol: Welcome to Apache Stanbol!* Apache, 2010-12-14. Web. 9th Aug. 2014.

'Stargazers'. *wooorm/retext*. GitHub, 8th Aug. 2014. Web. 8th Aug. 2014.

'Summly'. *Summly: Pocket sized news for iPhone*. Summly, 2012. Web. 8th Aug. 2014.

Suzuki, Yusuke [constellation]. 'escodegen'. *constellation/escodegen*. GitHub, 28th July 2014, version 1.3.3. Web. 9th Aug. 2014.

'TextRazor'. *TextRazor: The Natural Language Processing API*. TextRazor, 2014. Web. 8th Aug. 2014.

'The MIT License (MIT)'. *Open Source Initiative*. The Open Source Initiative, web. 8th Aug. 2014.

'Travis'. *Travis*. Travis CI GmbH, 2014. Web. 8th Aug. 2014.

U, Eric [ericu]. '[fileapi-directories-and-system/filewriter]'. W3C, 2nd Apr. 2014. Web. 8th Aug. 2014.

Umbel, Chris, Rob Ellis and Ken Koch. 'natural'. *NaturalNode/natural*. GitHub, 22nd June 2014, version 0.1.28. Web. 9th Aug. 2014.

Wormer, Titus [wooorm_]. 'DailyJS: Natural Language Parsing with Retext'. *JavaScript*. Reddit, 5th Aug. 2014. Web. 8th Aug. 2014.

———[wooorm_]. 'Natural Language Parsing with Retext'. *Node*. Reddit, 1st Aug. 2014. Web. 8th Aug. 2014.

Young, Alex. 'Natural Language Parsing with Retext'. *DailyJS: A JavaScript Blog*. 31st July 2014. Web. 8th Aug. 2014.

Zakas, Nicholas [eslint]. 'eslint'. *eslint/eslint*. GitHub, 6th Aug. 2014, version 0.7.4. Web. 9th Aug. 2014.

Zimmerman, Miles [mileszim]. 'sediment'. *mileszim/sediment*. GitHub, 9th Sept. 2013, version 0.9.2. Web. 9th Aug. 2014.