# TermSuite: Terminology Extraction with Term Variant Detection

**Damien Cram**
LINA - UMR CNRS 6241
Université de Nantes, France
damien.cram@univ-nantes.fr

**Béatrice Daille**
LINA - UMR CNRS 6241
Université de Nantes, France
beatrice.daille@univ-nantes.fr

## Abstract

We introduce, TermSuite, a JAVA and UIMA-based toolkit to build terminologies from corpora. TermSuite follows the classic two steps of terminology extraction tools, the identification of term candidates and their ranking, but implements new features. It is multilingually designed, scalable, and handles term variants. We focus on the main components: UIMA Tokens Regex for defining term and variant patterns over word annotations, and the grouping component for clustering terms and variants that works both at morphological and syntactic levels.

## 1 Introduction

Terminologies play a central role in any NLP applications such as information retrieval, information extraction, or ontology acquisition. A terminology is a coherent set of terms that constitutes the vocabulary of a domain. It also reflects the conceptual system of that domain. A term could be a single term (SWT), such as *rotor*, or a complex term. Complex terms are either compounds such as *broadband*, or multi-word terms (MWT) such as *frequency band*. Terms are functional classes of lexical items used in discourse, and as such they are subjected to linguistic variations such as modification or coordination.

As specialized domains are poorly covered by general dictionaries, Term Extraction Tools (TET) that extract terminology from corpora have been developed since the early nineties. This first generation of TET (Cabré et al., 2001) was monolingually designed, not scalable, and they were not handling term variants, except for ACABIT (Daille, 2001) and FASTR (Jacquemin, 2001). This last question has always been a pain in the neck for TET.

The current generation of TET improves on various aspects. As an example, TermoStat[1] deals with several Romance languages, reaches to treat text up to 30 megabytes, and proposes a first structuring based on lexical inclusion. TermSuite goes a step forward: it is multilingually designed, scalable, and handles term variants. It is able to perform term extraction from languages that behave differently from the linguistic point of view. Complex terms in languages such as German and Russian are mostly compounds, while in Roman languages they are MWT. TermSuite extracts single terms and any kind of complex terms. For some generic domains and some applications, large amounts of data have to be processed. TermSuite is scalable and has been applied to corpora of 1.1 gigabytes using a personal computer configuration. Finally, TermSuite identifies a broad range of term variants, from spelling to syntactic variants that may be used to structure the extracted terminology with various conceptual relations.

Since the first TermSuite release (Rocheteau and Daille, 2011), several enhancements about TET have been made. We developed UIMA Tokens Regex, a tool to define term and variant patterns using word annotations within the UIMA framework (Ferrucci and Lally, 2004) and a grouping tool to cluster terms and variants. Both tools are designed to treat in an uniform way all linguistic kinds of complex terms.

After a brief reminder of TermSuite general architecture, we present its term spotting tool UIMA Tokens Regex, its variant grouping tool, and the variant specifications we design for English, French, Spanish, German, and Russian. Fi-

[1] http://termostat.ling.umontreal.ca/



*Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics—System Demonstrations*, pages 13–18,
Berlin, Germany, August 7-12, 2016. 

nally, we provide some figures and considerations about TermSuite resources and behaviour.

## 2 TermSuite architecture

TET are dedicated to compute the termhood and the unithood of a term candidate (Kageura and Umino, 1996). Two steps make up the core of the terminology extraction process (Pazienza et al., 2005):

1. Spotting: Identification and collection of term-like units in the texts, mostly a subset of nominal phrases;
2. Filtering and sorting: Filtering of the extracted term-like units that may not be terms, syntactically or terminologically; Sorting of the term candidates according to their unithood, their terminological degree and their most interest for the target application.

TermSuite adopts these two steps. Term-like units are collected with the following NLP pipeline: tokenization, POS tagging, lemmatization, stemming, splitting, and MWT spotting with UIMA Tokens Regex. They are ranked according to the most popular termhood measure. But in order to improve the term extraction process and to provide a first structuring of the term candidates, a component dedicating to term variant recognition has been added. Indeed, term variant recognition improves the outputs of term extraction: the ranking of the term candidates is more accurate and more terms are detected (Daille and Blancafort, 2013).

Figure 2 shows the output of TermSuite TET within the graphical interface. The main window shows the terms rank according to termhood. A term candidate may group miscellaneous term variants. When a term is highlighted, the occurrences spot by UIMA Tokens Regex are showed in the bottom window and the term features in the right window.

## 3 Spotting multiword terms

We design a component in charge of spotting multi-word terms and their variants in text, which is based on UIMA Tokens Regex[2], a concise and expressive language coupled with an efficient rule engine. UIMA Tokens Regex allows the user to define rules over a sequence of UIMA annotations, *ie.* over tokens of the corpus, each rule being in the form of a regular expression. Compared to RUTA (Kluegl et al., 2016), UIMA Tokens Regex operates only on annotations that appear sequentially, which is the case for word annotations. The occurrence recognition engine has been thus implemented as a finite-state machine with linear complexity.

### 3.1 Syntax

UIMA Tokens Regex syntax is formally defined by an ANTLR[3] grammar and inspired by Stanford TokensRegex (Chang and Manning, 2014).

**Matchers** Before defining regular expressions over annotations, each annotation needs to be atomically matchable. That is why UIMA Tokens Regex defines a syntax for *matchers*. A *matcher* can be of three types:

| | |
|---|---|
| [*Boolean Exp*] | an expression matching the values of annotation attributes. |
| /*String RegExp*/ | A valid Java regular expression matching against the text covered by the annotation. |
| The dot "`.`" | matches any annotation. |

The *Boolean Exp* within brackets is a combination of *atomic boolean expressions*, boolean operators & and ||, and parentheses. An *atomic boolean expression* is of the form:

`property op literal`

Where `property` is an annotation feature defined in TermSuite UIMA type system, `op` is one of ==, !=, <, <=, >, and >=, and `literal` is either a string, a boolean (`true` or `false`), or a number (integer or double).

**Rules** *Rules* are named regular expressions that are defined as follows:

`term "rule name":` *TokensRegex*;

Where *TokensRegex* is a sequence of quantified *matchers*. The quantifiers are:

| | |
|---|---|
| `?` | 0 or 1 |
| `*` | 0 or several |
| `+` | at least 1 |
| `{n}` | exactly $n$ |
| `{m,n}` | between $m$ and $n$ |

[2] http://github.com/JuleStar/uima-tokens-regex/

[3] http://antlr.org/

### 3.2 Engine

UIMA Tokens Regex engine parses the list of rules and creates for each of these rules a finite-state automaton. The engine provides automata with the sequence of UIMA annotations of the preprocessed input document. UIMA Tokens Regex engine implements the default behaviour of a regular expression engine: it is *greedy*, *backtracking*, picking out the *first alternative*, and *impatient*.

Every time an automaton (*ie.* a *rule*) matches, `TermSuite` generates a rule *occurrence* and stores the offset indexes of the matched text.

### 3.3 Application to terminology extraction

**Example** In `TermSuite` type system, the values of the feature `category` are the part-of-speech (POS) tags. Rule `an` below extracts MWT composed of one or several adjectives followed by a noun.

```
term "an": [category=="adjective"]+
           [category=="noun"] ;
```

**Matcher predefinition** For the sake of both readability and reusability, UIMA Tokens Regex allows the user to predefine matchers. Thus, Rule `an` can be expressed concisely as `A+ N` using the matchers `N` and `A`:

```
matcher N: [category=="noun"];
matcher Vpp: [V & mood=="participle"
              & tense=="past"];
matcher A: [(Vpp | category=="adjective")
              & lemma!="same"
              & lemma!="other"];
matcher C: /^(and|or)$/;
matcher D: [category=="determiner"
         & subCategory != "possessive"];
matcher P: [category=="adposition"
        & subCategory=="preposition"];

term "an": A+ N ;
term "npn": N P D? N ;
term "acan": ~D A C A N ;
```

Rule `acan` extracts coordination variants that match the "adjective conjunction adjective noun" pattern, such as *onshore and offshore locations*. The quantifier `?` expresses an optional determiner. Rule `npn` can extract both MWT: *energy of wind* and *energy of the wind*.

**Features** The annotation features available in `TermSuite` type system are `category`, `subCategory`, `lemma`, and `stem` and inflectional features such as `mood`, `tense`, or `case`.

**Lexical filtering** Matcher `A` above shows an example of lexical filtering that prohibits occurrences of the listed lemma in the pattern. For example, Rule `an` will not match the term candidate *same energy*.

**Contextual filtering** Contextual POS are preceded by *tilde* ($\sim$). Rule `acan` shows an example of contextual filtering. A determinant should occur for the pattern to be matched, but it will be not part of collected MWT.

## 4 Variant grouping

`TermSuite` is able to gather terms according to syntactic and morphological variant patterns that are defined with YAML syntax (Ben-Kiki et al., 2005).

### 4.1 Syntax

A variant rule states a set of conditions that two term candidates must fulfil to be paired. It consists of:

**a rule name** a string expression between double quotes (`"`), ended by a colon (`:`),

**a source pattern** and **a target pattern**, which are sequences of `matcher` labels.

**a boolean expression** a logical expression on source and target term features, denoted by `rule`. The field `rule` is interpreted by a Groovy engine and must be defined in valid Groovy syntax.

**Example** The example below is the simplest variant grouping rule defined for English.

```
"S-I-NN-(N|A)":
  source: N N
  target: N N N, N A N
  rule: s[0]==t[0] && s[1]==t[2]
```

This rule is named `S-I-NN-(N|A)`. It states that one term candidate (the source) must be of pattern N N, and the second term candidate (the target) of patterns N N N or N A N. The `rule` field states that the `lemma` property of $s[0]$, the first noun of the *source*, has the same lemma as $t[0]$, the first noun of the `target`. Likewise $s[1]$ and $t[2]$ must share the same lemma. For example, this variant grouping rule will be satisfied for the two terms *turbine structure* and *turbine base structure*.

**Word features** The `rule` field expresses conditions on word features. The two main features used for grouping are `lemma` and `stem`. `lemma` is the default one, that is why stating $s[0] == t[0]$ is equivalent to $s[0].lemma == t[0].lemma$. The rule "S-PI-NN-P" below makes use of the `stem` property. An example of grouping is *effect of rotation* and *rotational effect* where *rotational* is derived from *rotation*.

```
"S-PI-NN-P":
  source: N P N
  target: A N, N N
  rule: s[0]==t[1] && s[2].stem==t[0].stem
```

**Morphological variants** `TermSuite` implements Compost, a multilingual splitter (Loginova Clouet and Daille, 2014) that makes the decision as to whether the term composed of one graphic unit, is a SWT or a compound, and for compounds, it gives one or several candidate analyses ranked by their scores. We only keep the best split. The compound elements are reachable when `TermSuite` comes to apply the variant grouping rules. The syntax of YAML variant rules allows the user to express morphological variants between two terms:

```
"M-I-EN-N|A":
  source: N [compound]
  target: N N, A N
  rule: s[0][0]==t[0][0] && s[0][1] == t[1]
```

In the rule `M-I-EN-N|A` above, the tag `[compound]` after the source pattern states that the source has to be a morphosyntactic compound. In the `rule` field, we access the component features with the second index of the two-based indexing arrays, the first index referring to the POS position in the source or target patterns. As examples, this rule groups the two term candidates *windfarm* and *windmill farm*, and also *hydropower* and *hydroelectric power*.

### 4.2 Engine

Term variant grouping applies on term pairs with a complexity of $O(n^2)$, where $n$ is the number of term candidates extracted by UIMA Tokens Regex. `TermSuite` copes with this issue by pre-indexing each term candidate with all its pairs of single-word lemmas. For example, the term of length 3 *offshore wind turbine* has three indexing keys: (`offshore`, `wind`), (`offshore`, `turbine`), and (`turbine`, `wind`). The grouping engine operates over all terms sharing the same indexing key, for all indexing keys. Therefore, the $O(n^2)$ complexity applies to small subsets of term candidates, and the weight of variant grouping in the overall terminology extraction process is quite reasonable (see Section 7).

| | MWT | Variants |
|---|---|---|
| en | 43 | 41 |
| fr | 35 | 37 |
| de | 20 | 30 |
| es | 62 | 40 |
| ru | 18 | 16 |

Table 1: Numbers of rules provided in `TermSuite`

## 5 Language grammars

We define MWT spotting rules and variant grouping rules for the five languages supported by `TermSuite`: Fr, En, Es, De, and Ru. Table 1 shows the number of rules by languages for MWT spotting and for term variant grouping.

## 6 Ranking by termhood

Term candidates are ranked according to their termhood that is measured with *weirdness ratio* (`WR`). `WR` is the quotient of the relative frequency in both the domain specific corpus $\mathcal{C}$ and a general language corpus $\mathcal{G}$.

$$WR(t, \mathcal{C}) = \frac{f_{norm}(t, \mathcal{C})}{f_{norm}(t, \mathcal{G})} \quad (1)$$

Where $f_{norm}$ stands for the normalized frequency of a term in a given corpus, *ie.* the average number of its occurrences every 1000 words, and $\mathcal{G}$ is a general language corpus.

### 6.1 General language corpus

The general language corpora used for computing `WR` are part of the compilation of newspapers provided by CLEF 2004 (Jones et al., 2005). These corpora cover numerous and miscellaneous topics, which are useful to obtain a corpus representative of the general language. The corpora of the general language that we use to compute the frequencies of term candidates are:

| Newspaper | Lang | Size | Nb words |
|---|---|---|---|
| Der Spiegel | De | $382M$ | 60M |
| Glasgow Herald | En | $302M$ | 28M |
| Agencia EFE | Es | $1.1G$ | 171M |
| Le Monde | Fr | $1.1G$ | 82M |
| Izvestia | Ru | $66M$ | 5.8M |

### 6.2 WR behaviour

Figure 1 gives WR distribution on the English part of the domain-specific monolingual comparable corpora for Wind Energy[4] [EOL]. [EOL] is available for seven languages and has a minimum size of $330K$ words by language. The x-axis of Figure 1 is set to WR base-10 logarithm, hence a value of 2 means that the term candidate is a 100 times more frequent in the specific corpus $\mathcal{C}$ than in $\mathcal{G}$.

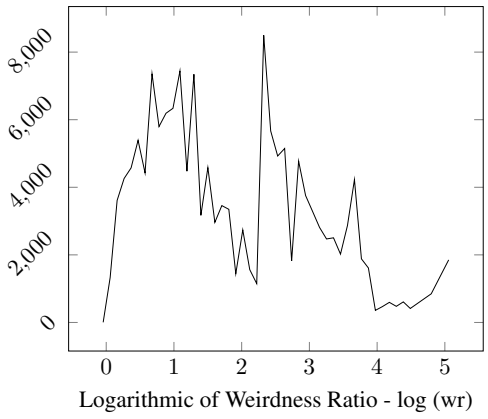


Figure 1: Distribution of WR base-10 logarithm over all terms extracted by TermSuite on English [EOL].

We distinguish two sets of terms on Figure 1. The first one, starting around 0 until $log(wr) \simeq 2$, contains the terms that are not domain specific since they occur in both the specialised and the general language corpora. The second set, from the peak at $log(wr) \simeq 2$ to the upper bound, contains both the terms that appear much more frequently in $\mathcal{C}$ than in $\mathcal{G}$ and the terms that never occur in $\mathcal{G}$. Actually, the first peak at $log(wr) \simeq 2$ refers to terms that occur once in $\mathcal{C}$ and never in $\mathcal{G}$, the second lower peak refers to terms that occur twice in $\mathcal{C}$ and never in $\mathcal{G}$, and so on.

We did not provide the distributions for other [EOL] languages nor for other corpora, because their WR distributions are similar. For all configurations, the first peak always appears at WR $\simeq 2$ and the upper bound at WR $\simeq 5$. As a result of the analysis of WR distribution, we set 2 as default value of $log(wr)$ threshold for accepting candidates as terms.

[4] http://www.lina.univ-nantes.fr/taln/maven/wind-energy.tgz

## 7 Performances

TermSuite operates on English [EOL] in 11 seconds with the technical configuration: Ubuntu 14.04, 16Go RAM, Intel(R) Core(TM) i7-4800MQ (4x2,2.7Ghz). We detail the execution times of each main component with the use of two part-of-speech taggers TreeTagger[5](TT) and Mate[6]:

| | TT | Mate |
|---|---|---|
| Tokenizer | 1.3s | *idem* |
| POS/Lemmatiser | 2.4s | 81s |
| Stemmer | 0.67s | *idem* |
| MWT Spotter | 4.8s | *idem* |
| Morph. Compound Finder | 0.14s | *idem* |
| Syntactic Term Gatherer | 0.23s | *idem* |
| Graphical Term Gatherer | 0.27s | *idem* |
| Total (without UIMA overheads) | 9.8s | 88.5s |

**Scalability** Time complexity is linear. The processing of Agencia EFE corpus (*cf.* Section 6.1), the biggest tested so far (171 million words), takes 101 minutes to process. This performance proves a very satisfactory vertical scalability in the context of smaller domain-specific corpora. No kind of parallelism has been implemented so far, not even Java multi-threading, which is the best opportunity of optimization if an improvement of performances is required.

## 8 Release

TermSuite is a Java (7+) program. It can be used in three ways: the Java API, the command line API, or the graphical user interface as shown on Figure 2. Its only third-party dependency is TreeTagger, which needs to be installed separately and referenced by TermSuite configuration.

TermSuite is under licence Apache 2.0. The source code and all its components and linguistic resources are released on Github[7]. The latest released versions, currently 2.1, are available on Maven Central[8]. All links, documentation, resources, and guides about TermSuite are available on its official website:

http://termsuite.github.io/

## Acknowledgements

TermSuite development is supported by ISTEX, French Excellence Initiative of Scientific

[5] http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

[6] https://code.google.com/p/mate-tools/

[7] https://github.com/termsuite/

[8] Maven group id is fr.univ-nantes.termsuite

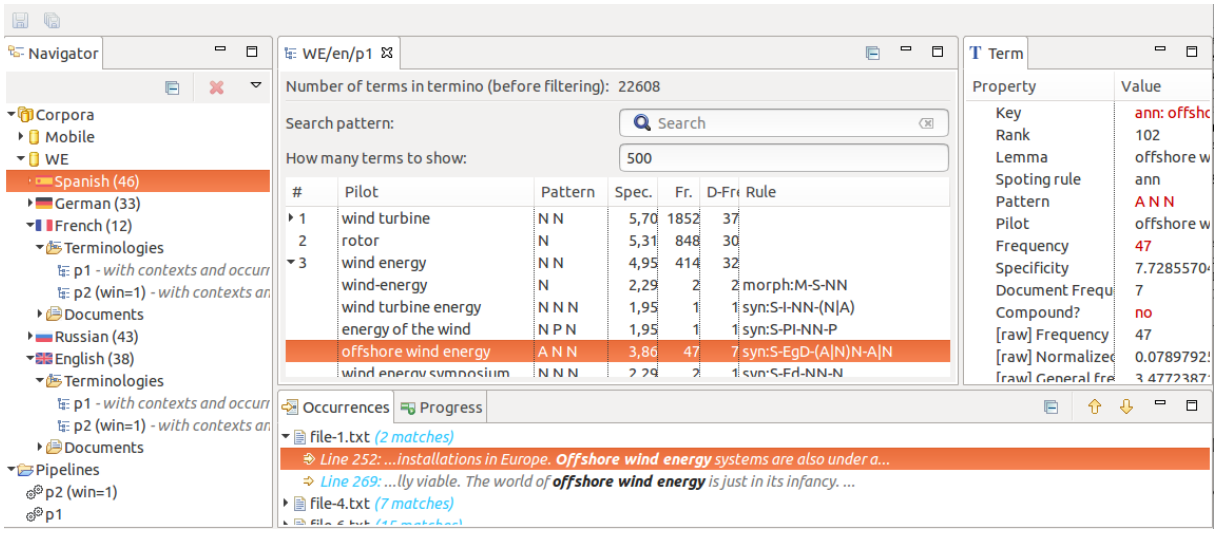


Figure 2: `TermSuite` graphical user interface

and Technical Information.

## References


Oren Ben-Kiki, Clark Evans, and Brian Ingerson. 2005. Yaml ain't markup language (yaml™) version 1.1. *yaml. org, Tech. Rep*.

M. Teresa Cabré, Rosa Estopà Bagot, and Jordi Vivaldi Platresi. 2001. Automatic term detection: A review of current systems. In D. Bourigault, C. Jacquemin, and M.-C. L'Homme, editors, *Recent Advances in Computational Terminology*, volume 2 of *Natural Language Processing*, pages 53–88. John Benjamins.

Angel X. Chang and Christopher D. Manning. 2014. TokensRegex: Defining cascaded regular expressions over tokens. Technical Report CSTR 2014-02, Department of Computer Science, Stanford University.

Béatrice Daille and Helena Blancafort. 2013. Knowledge-poor and knowledge-rich approaches for multilingual terminology extraction. In *Proceedings, 13th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, page 14p, Samos, Greece.

Béatrice Daille. 2001. Qualitative terminology extraction. In D. Bourigault, C. Jacquemin, and M.-C. L'Homme, editors, *Recent Advances in Computational Terminology*, volume 2 of *Natural Language Processing*, pages 149–166. John Benjamins.

David Ferrucci and Adam Lally. 2004. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10:327–348.

Christian Jacquemin. 2001. *Spotting and Discovering Terms through Natural Language Processing*. Cambridge: MIT Press.

Gareth J. F. Jones, Michael Burke, John Judge, Anna Khasin, Adenike Lam-Adesina, and Joachim Wagner, 2005. *Multilingual Information Access for Text, Speech and Images: 5th Workshop of the Cross-Language Evaluation Forum, CLEF 2004, Bath, UK, September 15-17, 2004, Revised Selected Papers*, chapter Dublin City University at CLEF 2004: Experiments in Monolingual, Bilingual and Multilingual Retrieval, pages 207–220. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kyo Kageura and Bin Umino. 1996. Methods of automatic term recognition: a review. *Terminology*, 3(2):259–289.

Peter Kluegl, Martin Toepfer, Philip-Daniel Beck, Georg Fette, and Frank Puppe. 2016. UIMA ruta: Rapid development of rule-based information extraction applications. *Natural Language Engineering*, 22(1):1–40.

Elizaveta Loginova Clouet and Béatrice Daille. 2014. Splitting of Compound Terms in non-Prototypical Compounding Languages. In *Workshop on Computational Approaches to Compound Analysis, COLING 2014*, pages 11 – 19, Dublin, Ireland, August.

Maria Teresa Pazienza, Marco Pennacchiotti, and Fabio Massimo Zanzotto. 2005. Terminology extraction: An analysis of linguistic and statistical approaches. In S. Sirmakessis, editor, *Proceedings of the NEMIS 2004 Final Conference*, volume 185 of *Studies in Fuzziness and Soft Computing*, pages 225–279. Springer Berlin Heidelberg.

J. Rocheteau and B. Daille. 2011. TTC TermSuite - A UIMA Application for Multilingual Terminology Extraction from Comparable Corpora. In *Proceedings of the 5th International Joint Conference on Natural Language Processing (IJCNLP 2011*, Thailand, November. Asian Federation of ACL.