Valencer: an API to Query Valence Patterns in FrameNet

Alexandre Kabbach and **Corentin Ribeyre**
University of Geneva
Department of Linguistics
5 Rue de Candolle, CH-1211 Genève 4
firstname.lastname@unige.ch

Abstract

This paper introduces `Valencer`: a RESTful API to search for annotated sentences matching a given combination of syntactic realizations of the arguments of a predicate – also called *valence pattern* – in the FrameNet database. The API takes as input an HTTP `GET` request specifying a valence pattern and outputs a list of exemplifying annotated sentences in JSON format. The API is designed to be modular and language-independent, and can therefore be easily integrated to other (NLP) server-side or client-side applications, as well as non-English FrameNet projects.

1 Introduction

The Berkeley FrameNet project (Baker et al., 1998) aims at creating a human and machine-readable lexical database of English, supported by corpus evidence annotated in terms of *frame semantics* (Fillmore, 1982). Its output takes the form of a database of corpus-extracted and annotated sentences specifying schematic representations of events, relations or entities called *frames*, frame-evoking words called *lexical units*, and semantic roles called *frame elements*. The latest data release contains about 1,200 frames, 10,000 frame elements, 13,000 lexical units and 200,000 manually annotated sentences (see Section 2 for examples of FrameNet annotated sentences).

Computational linguistics applications such as information extraction (Surdeanu et al., 2003), paraphrase recognition (Padó and Erk, 2005), question answering (Shen and Lapata, 2007) and parsing (Das et al., 2013) have made extensive use of FrameNet taxonomy and its documentation of the syntactic valence of the arguments of predicates. Information regarding predicate-argument structures – referred to as *valence patterns* in FrameNet (see Section 2) – could also benefit corpus linguists, alongside NLP applications, when searching for complex semantic and/or syntactic patterns not bounded by given lexical items, overcoming thereby the limitations of traditional concordancers (Manning, 2003).

However, given the current structure of FrameNet data (Baker et al., 2003), valence patterns cannot be searched directly and can only be accessed through the lexical units they refer to, although a given valence pattern may be realized in multiple lexical units across several distinct frames. Therefore, searching for all annotated sentences matching a given valence pattern, across lexical units and frames, requires some additional pre-processing of FrameNet data, beside the implementation of a specific search engine.

In this paper we address this issue and introduce `Valencer`: a RESTful API to search for annotated sentences matching a given valence pattern in the FrameNet database. The API takes as input an HTTP `GET` request specifying the queried valence pattern (see Section 3.3) and outputs a list of annotated sentences in JSON format (see Section 3.4). The `Valencer` API provides a lightweight server-side application compatible with modern W3C standards. It removes from potential users the burden of having to import and index FrameNet data, validate input queries and optimize the valence pattern search engine. Its JSON output, consistent with FrameNet data structure, makes the API easy to integrate into other (NLP) server-side or client-side applications. Finally, being language-independent, the API can be smoothly adapted to other FrameNet projects (e.g. Japanese (Ohara et al., 2004)), if they use the same XML data release format as the Berkeley FrameNet. `Valencer` is open-source, licensed under the MIT license and freely available at `https://github.com/akb89/valencer`.

Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations, pages 156–160, Osaka, Japan, December 11-17 2016.

2 Valence Patterns in FrameNet

In FrameNet, syntactic realizations of frame elements are called *valences* and are represented as triplets FE.PT.GF of *frame element* (FE), *phrase type* (PT) and *grammatical function* (GF). *Valence patterns* refer to the range of combinatorial possibilities of valences for each lexical unit. Examples of valence patterns are given in (1) and (2) for the lexical unit *give*.v in the `Giving` frame. The two valence patterns differ in the morpho-syntactic realizations of their THEME and RECIPIENT frame elements.

(1) a. He gives local charities money

b. [He]$_{\text{Donor.NP.Ext}}$ **gives** [local charities]$_{\text{Recipient.NP.Obj}}$ [money]$_{\text{Theme.NP.Dep}}$

(2) a. He gives money to local charities

b. [He]$_{\text{Donor.NP.Ext}}$ **gives** [money]$_{\text{Theme.NP.Obj}}$ [to local charities]$_{\text{Recipient.PP[to].Dep}}$

In (1) and (2), 'NP' refers to a *noun phrase*, 'PP[to]' to a *prepositional phrase* headed by *to*, 'Ext' to an *external argument* (the subject), 'Obj' to an *object* and 'Dep' to a *dependent*.

3 API overview

3.1 Architecture

The `Valencer` API is a JavaScript Node.js-based RESTful web application relying on a MongoDB database. The workflow of the API follows: (1) receive an HTTP `GET` request specifying a valence pattern, (2) validate the query and its parameters, (3) retrieve and return the relevant data, and (4) output a collection of documents in JSON format. The output documents correspond to populated MongoDB entries. The technological choice of a document-based, JSON-oriented NoSQL database such as MongoDB is particularly relevant to our case as it allows us to keep consistency between the structure of the data output by the API and the structure of the data stored in the database. Additionally, the JSON format of the output makes the API particularly well-suited for integration with JavaScript web clients.

3.2 Underlying technologies

The choice of JavaScript, Node.js and MongoDB is primarily motivated by considerations of performance and maintainability. Performances of the V8 engine powering Node.js have turned JavaScript into a serious challenger of PHP for server-side technologies, especially when PHP is not used with a JIT compiler. Additionally, JavaScript asynchronous programming, especially when implemented with the async/await features of ECMAScript 2017, brings the benefits of concurrent programming without the traditional shortcomings of callbacks (see `http://callbackhell.com/`). It may even yield performance gains over traditional multi-threading approaches while avoiding complexity overhead. Finally, a multi-purpose technological environment, coded in a single programming language and able to handle both back-end and front-end computing as well as (XML) datasets imports greatly decreases refactoring and debugging complexity and improves long-term maintainability. Moreover, schemaless databases such as MongoDB provide a flexible architecture for handling sparse data, easy manipulation of complex tree-structures, and a seamless mapping to human-readable XML formats.

3.3 Input

The API takes as an entry point an HTTP `GET` request specifying a valence pattern 'vp'. For example, the query corresponding to the valence pattern in sentence (2b) is:

```
GET/annoSets?vp=Donor.NP.Ext Theme.NP.Obj Recipient.PP[to].Dep
```

The API is flexible and can process combinations of triplets FE.PT.GF in any order (e.g. PT.FE.GF, GF.PT.FE). It can also process partial triplets, with up to two non-specified elements (FE.PT, GF, PT.GF, etc.). This enables the API to process "semantic queries" – queries specifying only frame elements – such as `Donor Theme Recipient`, as well as "syntactic queries" – queries specifying only phrase types and/or grammatical functions – such as `NP.Ext NP.Obj PP[to].Dep`, and, of course, arbitrary combinations of both, such as `NP.Ext Theme Recipient.PP[to]`.

annotationSet document part 1/2


```
{"annotationSet": {
        "_id": 1632555,
        "sentence": {
                "_id": 1090710,
                "text": "He gives money to local
                    charities . ",
                ...
        },
        "lexUnit": {
                "_id": 4344,
                "name": "give.v",
        "frame": {
                "_id": 139,
                "name": "Giving",
                "lexUnits": [{
                        "_id": 4344,
                        "name": "give.v"}, {
                        "_id": 5344,
                        "name": "donate.v"}, {
                        ...
                }],
                "frameElements": [{
                        "_id": 1052,
                        "name": "Donor",
                        ...}, {
                ...
                }],
                ...
        },
        ...
```


annotationSet document part 2/2


```
        ...
        "pattern": {
                "_id": "57fc94026cc52246ae399541",
                "valenceUnits": [{
                        "_id": "57fc94026cc52246ae",
                        "FE": "Donor",
                        "PT": "NP",
                        "GF": "Ext"}, {
                        ...
                }]
        },
        "labels": [{
                "_id": "57fc94f96cc52246ae46e9ff",
                "name": "Donor",
                "type": "FE",
                "startPos": 0,
                "endPos": 1}, {
                "_id": "57fc94f96cc52246ae46ea05",
                "name": "NP",
                "type": "PT",
                "startPos": 0,
                "endPos": 1}, {
                "_id": "57fc94f96cc52246ae46ea02",
                "name": "Ext",
                "type": "GF",
                "startPos": 0,
                "endPos": 1}, {
                ...
        }]
}}
```


Figure 1: A sample output of the `Valencer` API: the annotationSet document corresponding to sentence (2a) "He gives money to local charities". For readability, the document is split into two parts.

3.4 Output

The `Valencer`API is primarily designed to output a collection of *annotationSet* documents (see Figure 1). In the original FrameNet XML data, annotationSet tags are found under two separate subgroups of the lexical unit entities: they connect the part which lists the syntactic realization of the arguments of the predicate (the valence patterns of the lexical unit) to the part which lists the annotated sentences exemplifying each valence pattern and their respective labels. In the `Valencer`, the annotationSet object merges all this information into one object: it centralizes information regarding a specific annotated sentence, its label, the lexical unit it refers to and the specific valence pattern it exemplifies. All original FrameNet ids are kept to potentially retrieve the original entities directly into the FrameNet database.

3.5 Authentication

The `Valencer` API follows a traditional HMAC-SHA1 key/secret authentication process to allow access to the API methods. The header of each HTTP request to the API must include a key, a Unix timestamp and a signature. The signature itself is the concatenation of the specified API route, the specified query and the Unix timestamp. It is hashed using a SHA1 algorithm and the secret corresponding to the key. At each HTTP request, the server recomputes the signature using the stored secret corresponding to the specified key and checks if it matches the signature passed to the header before accepting or rejecting the request. The timestamp is used to prevent man-in-the-middle attacks by setting a validation period for querys, disallowing thereby replay attacks using stolen keys, querys, and signatures.

4 Use Cases

By design the `Valencer` API is primarily intended to be integrated to other NLP systems or plugged to a web-based client, which is why all necessary information regarding an annotated sentence are gathered in a single annotationSet. However, to a human user, an annotationSet may include a lot of irrelevant information, such as object ids or references, which may render the analysis of the output of the API rather tedious. In order to better illustrate the functionalities of the `Valencer` API, we have implemented four

additional routes in our middlewares, beside GET/annoSets, to extract and process only the necessary attributes of an annotationSet entity depending on specific use cases.

4.1 Get Lexical Units

GET/lexUnits returns a collection of lexical units, with their respective names and frame names, which contain at least one reference to the specified valence pattern given in input. It can be helpful, e.g., in searching for paraphrasing candidates, as FrameNet is characterized by relatively narrow-scope frames and frame elements. Indeed, by definition, lexical units sharing specific valence patterns should be relatively close semantically. For example, querying for the valence pattern Donor.NP.Ext Theme.NP.Obj Recipient.PP[to].Dep corresponding to sentence (2b) returns eleven lexical units, ten of which are in the Giving frame: *bequeath*.v, *contribute*.v, *donate*.v, *gift*.v, *give out*.v, *give*.v, *hand in*.v, *hand out*.v, *hand over*.v and *hand*.v. All verbs should therefore form valid sentences relatively close in meaning when replacing the verb *give* in sentence (2a): "He gives money to local charities". GET/lexUnits can also be used to analyze the "semantic scope" of a specific (syntactic) construction, by checking which lexical units match a given "syntactic" valence pattern, i.e., a valence pattern with unspecified frame elements. Querying, for instance, for the pattern NP.Ext NP.Obj NP.Dep corresponding to a prepositional indirect object construction returns a list of 346 unique lexical units (from a total of about 13,000) found in 206 frames (from a total of about 1,200).

4.2 Get Frames

GET/frames returns a collection of unique frame names corresponding to frames which contain lexical units which themselves contain at least one reference to the specified valence pattern given in input. Similarly to GET/lexUnits, GET/frames can be used to investigate the semantic scope of a given valence pattern (see Section 4.1). Additionally, GET/frames can be used to check which frames a frame element belongs to, an information that is not straightforwardly available in FrameNet (one has to search through all related frames to check whether or not it contains the frame element). Due to the diversity of semantic relations between frames in FrameNet – referred to as *frame relations* – frame elements can appear in more than one frame, and some (relatively abstract) frame elements can even appear in a significant number of frames. For instance, the DONOR and THEME frame elements of example (1) and (2) appear in 3 and 60 frames respectively.

4.3 Get Patterns

GET/patterns returns a collection of (valence) patterns – itself a collection of valenceUnit objects with FE, PT, GF attributes – matching the input. It is mostly useful for checking with which other valence units a given valence unit is realized. For example, querying for Donor.NP.Ext returns 81 unique patterns with 127 exemplifying sentences. There are currently 54,264 unique valence patterns in the FrameNet database.

4.4 Get Valence Units

GET/valenceUnits returns a collection of unique valenceUnit objects matching the input. It is particularly useful for checking all the syntactic realizations of a given frame element, or all the frame elements realized in a given syntactic valence. For example, querying for the frame element DONOR returns 12 unique valence units such as Donor.PP[from].Dep or Donor.PP[of].Ext. Querying for PP[of].Ext returns a list of 16 valence units including frame elements such as DONOR, TOPIC, MESSAGE or ENTITY. Querying then back for the output valence units with GET/patterns, GET/frames or GET/lexUnits provides more information about each pattern, frames and lexical units in which the valence units are realized.

5 Related Work

It is already possible to search for complex syntactic constructions within treebanks using tools such as TGrep2[1]. The main benefit of using FrameNet instead of treebanks lays in the theoretical background of *frame semantics* situated at the interface between syntax and semantics. It makes it possible to incorporate semantics and search for complex combinations of both syntactic *and* semantic constructions (see Section 3.3). FrameNet also brings a fine-grained classification of frames and frame elements, a strong advantage over PropBank (Palmer et al., 2005) for tasks such as paraphrase generation (see Section 4.1). Finally, FrameNet is free and machine readable, contrary to VDE (Herbst et al., 2004), theoretically noise-free as manually annotated, contrary to VALEX (Korhonen et al., 2006), and has a broader coverage than VerbNet (Schuler, 2005).

6 Conclusion

This paper introduced `Valencer`: a free, open-source and language-independent RESTful API to enable querying for valence patterns in the FrameNet database. The `Valencer` renders parts of FrameNet data more straightforwardly accessible and can also prove useful in non-FrameNet-specific tasks such as searching for complex semantic and syntactic constructions or generating high quality paraphrase.

References

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet Project. In *Proceedings of ACL-COLING*, pages 86–90, Montréal, Québec, Canada, August. Association for Computational Linguistics.

Collin F. Baker, Charles J. Fillmore, and Beau Cronin. 2003. The Structure of the FrameNet Database. *International Journal of Lexicography*, 16(3):281–296.

Dipanjan Das, Desai Chen, André F. T. Martins, Nathan Schneider, and Noah A. Smith. 2013. Frame-semantic parsing. *Computational Linguistics*, 40(1):9–56.

Charles J. Fillmore, 1982. *Frame semantics*, pages 111–137. Hanshin Publishing Co., Seoul, South Korea.

Thomas Herbst, David Heath, Ian F. Roe, and Dieter Götz. 2004. *A Valency Dictionary of English: A Corpus-Based Analysis of the Complementation Patterns of English Verbs, Nouns and Adjectives*, volume 40. Mouton de Gruyter.

Anna Korhonen, Yuval Krymolowski, and Ted Briscoe. 2006. A Large Subcategorization Lexicon for Natural Language Processing Applications. In *Proceedings of LREC*, volume 6.

Christopher D. Manning. 2003. Probabilistic Syntax. In Rens Bod, Jennifer Hay, and Stefanie Jannedy, editors, *Probabilistic Linguistics*, chapter 8. MIT Press.

Kyoko Hirose Ohara, Seiko Fujii, Toshio Ohori, Ryoko Suzuki, Hiroaki Saito, and Shun Ishizaki. 2004. The Japanese FrameNet Project: An Introduction. In *Proceedings of the Workshop on Building Lexical Resources from Semantically Annotated Corpora at LREC*, pages 9–11.

Sebastian Padó and Katrin Erk. 2005. To Cause Or Not To Cause: Cross-Lingual Semantic Matching for Paraphrase Modelling. In *Proceedings of the Cross-Language Knowledge Induction Workshop*.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, 31(1):71–106.

Karin Kipper Schuler. 2005. *VerbNet: A broad-coverage, comprehensive verb lexicon*. Ph.D. thesis, University of Pennsylvania.

Dan Shen and Mirella Lapata. 2007. Using Semantic Roles to Improve Question Answering. In *Proceedings of EMNLP-CoNLL*, pages 12–21, Prague, Czech Republic, June. Association for Computational Linguistics.

Mihai Surdeanu, Sanda Harabagiu, John Williams, and Paul Aarseth. 2003. Using Predicate-Argument Structures for Information Extraction. In *Proceedings of ACL*, pages 8–15, Sapporo, Japan, July. Association for Computational Linguistics.

[1] `https://tedlab.mit.edu/~dr/Tgrep2/`