Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 7009–7014
Marseille, 11–16 May 2020

Word at a Glance:

Modular Word Profile Aggregator

Tomáš Machálek
Institute of the Czech National Corpus, Faculty of Arts - Charles University
nám. Jana Palacha 1/2, 116 38 Praha 1, Czech Republic
tomas.machalek@ff.cuni.cz

Abstract

Word at a Glance (WaG) is a word profile aggregator that provides means for exploring individual words, their comparison and translation, based on existing language resources and related software services. It is designed as a building kit-like application that fetches data from different sources and compiles them into a single, comprehensible and structured web page. WaG can be easily configured to support many tasks, but in general, it is intended to be used not only by language experts but also the general public.

Keywords: meta search, word profile, web application, user interface, research portal, language resources promotion

1. Introduction

Individual language research centres and other language infrastructures all over the world have already collected a vast amount of diverse, well organized language resources (LRs) for many languages. However, such a diversity of data sets and services can also be overwhelming and hard to navigate through for many users. Our experience in the Czech National Corpus (CNC) shows that it is helpful to point users towards a careful selection of corpora, as well as to offer use cases with appropriate reasoning and interpretation of the observed language phenomena.

In this paper, we introduce Word at a Glance (WaG) that has been created to address the issue. WaG is a web application that offers a brief profile of a query (typically a single word) that is attractive, user-friendly and easy to understand. It is designed as a set of customizable tiles to showcase relevant characteristics of a given word (or phrase) that can be derived from available data and presented to users (visualizations of metadata variations, development trends, collocations, translation equivalents, variation of individual forms in a paradigm based on real data etc.).

Several alternative approaches that served as an inspiration for WaG should be mentioned. Most prominently, there is the DWDS online dictionary (Klein and Geyken, 2010) that is very similar but lexically oriented. Another set of ideas comes from English-Corpora.org (Davies, 2008) with its corpus-based overall statistics, as well as Sketch Engine (Kilgarriff et al., 2014) with its collocation profiles. However, these approaches are not directly usable either because the interfaces are tailor-made for particular data or simply because of licensing reasons. In contrast, WaG is an open-source application created with reusability in mind. A typical layout of WaG can be seen in Figure 1, the production version is available at `https://www.korpus.cz/wag/`.

Figure 1: An overview of WaG user interface

2. Basic characteristics

WaG is a web application accessible either via a desktop web browser or a smart mobile device where it adapts its user interface for mobile-specific input and screen. It focuses on aggregating and compiling information out of multiple local or remote LRs and its presentation to the user.

The key strength of WaG lies in its full configurability. When installed, a provider of a WaG-based website/portal starts with an "empty board" where different visualizations connected to specific LRs can be put and arranged into a wide range of possible contents.

In this way, WaG can be configured e.g. as a pure lexicological tool or as an interface for searching within a digitized library. It can be run as a support tool for people learning

foreign languages or as a page for side-by-side comparison of results provided by different LRs.
It should be emphasized that WaG as a software is not just a platform for running these data presentations but also a set of ready-to-be-used components covering different areas of linguistic/humanities research (see Chapter 4.2.).

3. Target audience

WaG is intended to be used by both experienced researchers and general public users without any prior experience with linguistics and related tools. We are aware of the fact that possible use cases may differ significantly within such a diverse audience.
When considering an *experienced researcher*, WaG is expected to serve as a research starting point where it acts as a meta-search tool or *portal* providing broad overview of different word properties and allowing easy access to the original services/websites which provided respective information. In this sense, WaG can be also seen as a tool for LRs promotion.
For the *general public*, WaG should be able to provide easy to use tools helping users with their common language-related problems and also presenting them interesting facts about languages. For such a use case, a LRs provider cannot expect the users to deal with advanced (and sometimes quite crude) tools.

4. Architecture

4.1. Operation modes

Based on our analysis and on the experience with our other applications, we have been able to recognize *three distinct operating modes* which, as we believe, cover most important use cases one may encounter when performing word-based search:

- single word search within a single language,
- two or more words comparison within a single language (Figure 2),
- word translation (two languages).

Each of the modes can be configured for one or more primary languages which means the search within WaG is generally meant as a *search within a language* where different modules may use different corpora and other resources to fulfill the task. In addition, the translation mode is expected to have also at least one target language configured.

4.2. Tile

Tile is the basic building block of the WaG. It can be seen as an *extraction of a specific functionality and data out of an existing application*. Visually, it occupies a part of WaG result page (see Chapter 4.3.) where it transforms a response from a configured service into a visualisation which is typically a chart, word cloud, tabular data, structured text etc.
Furthermore, a tile may offer an *alternative view mode* which typically means table-formatted data as an alternative to some more graphical presentation form.

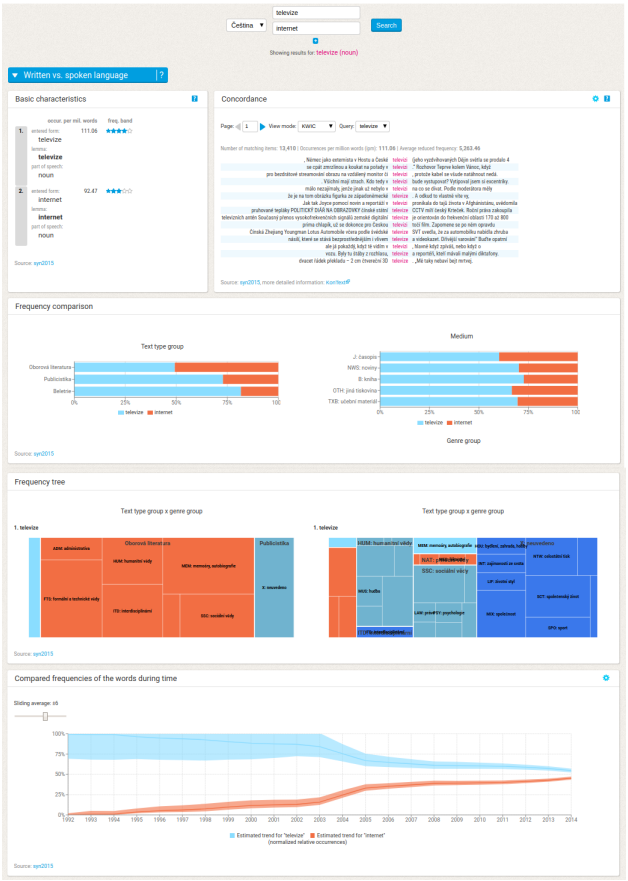

Figure 2: An example of a result page in the *word comparison mode*. Set-up like this can be applied e.g. in discourse analysis.

A tile may also offer end-user interactive configuration of its parameters (e.g. a search range for collocation calculation, min. frequency in a frequency distribution).
It is also possible to define a *backlink* to the original application which serves as a source of data/functionality for the tile. This property is important for the ”portal-like” behavior of WaG.
In general, there are no restrictions on which function and which data a tile may present. However, WaG prepackaged tiles cover the most typical corpus-based research topics (as viewed by CNC and its users). Most prominently, we should mention the CNC's database of translation equivalents Treq (Škrabal and Vavřín, 2017), an application for the study of competing language variants called SyD (Cvrček and Vondřička, 2011) and also KonText (Machálek, in press) as a provider of concordance-related search and analysis functions.
In terms of data resources, it is preferred if every tile type can be configured to fetch data from several alternative LRs or APIs of similar functionality. For WaG-prepackaged tiles, this means they typically rely on some abstract programming interface which is then implemented by different data services (see Chapter 4.5.). For example the *WordSimTile* (word similarity) is already able to load results from *Datamuse.com* service, from *CNC's internal word2vec service* or from the *REST API of the Leipzig Corpora Collection*. As another example, the *ConcordanceTile*

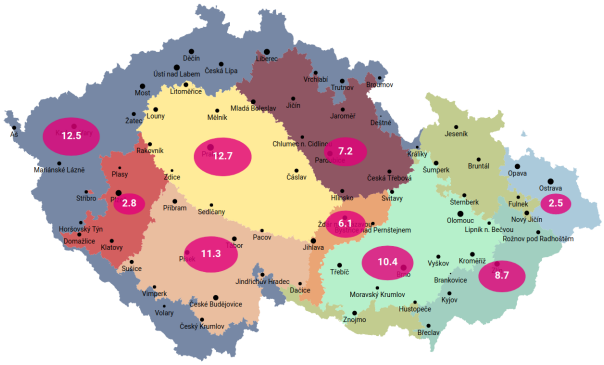

Figure 3: An example of geographically mapped frequency information

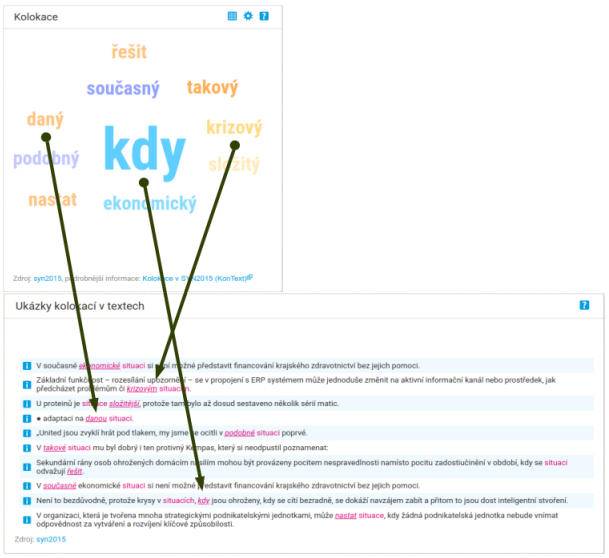

Figure 4: An example of a tile with results dependent on other tile

can be configured to read data from KonText or via the CLARIN Federated Content Search service.
The following WaG-prepackaged tiles are currently available:

- basic word profile (lemmatized variant, part of speech, relative frequency, frequency band, words with similar frequency),
- concordance (including parallel variant for two aligned languages),
- concordance merged from multiple filtered subsets of a source concordance,
- collocation profile,
- existing word forms of a given lemma,
- words with similar usage patterns (based on word embeddings),
- general pie chart and bar chart for text metadata frequency information,
- multi-source frequency bar chart (e.g. data merged from two or more different corpora),
- geographic area information mapped to an SVG image (Figure 3),
- development over time of a metadata attribute (e.g. publication date),
- general word translation and side-by-side comparable word translations based on multiple data sources,
- translation differences when restricted to specific data sources (e.g. fiction vs. newspapers),
- matching documents (list of documents with the highest occurrence of a searched word).

Some of those tiles have their counterparts in the *word comparison* mode, some tiles are able to operate in multiple modes themselves.
An example of an output provided by a production-ready instance can be seen in Figure 1 which shows a profile of the Czech word ”proces” (process). From user's point of view, there are the most frequent word forms and collocations with examples, as well as the distribution of ”proces” across the main text registers (where non-fiction largely prevails). This is supplemented by the frequency development across time. From technical point of view, there are various types of tiles employed that combine several different LRs in order to get an accurate picture of the actual use of this word as it is reflected in the CNC corpora.

4.3. Layout

While a configured tile can be seen as a ready-to-be-used building block, a layout can be seen as a tile installation frame.
On a desktop computer, the layout is composed of $N \times 3$ (rows $\times$ columns) cells where each tile may declare its desired width (1, 2 or 3 cells). More sophisticated tiles may offer an extra visual content when put across 2 or 3 cells. On a smartphone, the layout typically switches to $N \times 1$ grid.
To provide easy navigation through presented results, tiles are required to be organized into groups where each group has its heading and optional description (Figures 1 and 6).
Because of their different nature, each of the three operation modes (4.1.) must have its own layout configuration. Yet it is possible to omit some of the modes completely, in which case, users won't be able to select the disabled mode.

4.4. Tile dependency and interaction

Bringing results from different sources together and normalizing them is undoubtedly useful by itself. In addition, WaG offers also a mechanism for *interconnection of the individual tiles*.
Thinking about result types of *some* tiles, we can see that they produce basically *lists of words*. E.g. if we search for word collocations, similar words or word forms, we typically obtain a list of words probably along with a score information.

```
"CollocExamplesSynchronic": {
  "tileType": "ConcFilterTile",
  "label": {"cs-CZ": "Ukázky kolokací", "en-US": "Collocation examples"},
  "readSubqFrom": ["CollocationsSynchronic"],
  "apiURL": "http://kontext.korpus.test/kontext-api/quick_filter",
  "apiType": "kontext"
  "corpname": "syn2015",
  "posAttrs": ["word"],
  "metadataAttrs": [
    {"value": "doc.title", "label": {"cs-CZ": "Název", "en-US": "Title"}},
    {"value": "doc.author", "label": {"cs-CZ": "Autor", "en-US": "Author"}},
    {"value": "doc.biblio", "label": {"cs-CZ": "Bib. info", "en-US": "Bib. info"}}
  ]
}
```


Figure 5: A tile configuration example

A tile T_1 producing a list of words as a result, can be used in WaG as a source of so called *sub-query*. Once a tile T_2 is configured to listen to such a sub-query, it can base its result not just on the original user query but also on the sub-query provided by the T_1. So for example besides producing collocations of a word we can easily provide concordances containing examples of those collocations (Figure 4). Or we can extend user search by looking for synonyms and then looking for collocations of all the similar words.

4.5. Data service

Data service is an abstraction that interconnects WaG with various kinds of external data and remote services. In WaG, it is the client-side (i.e. a web browser) which deals with the data services. This predetermines WaG as an aggregator of services accessible via HTTP protocol which in our experience covers most of the published LRs and services today.

From a more technical point of view, data service is typically a simple module wrapping some HTTP API and transforming its response to a normalized form suitable for use by WaG and prescribed by a respective programming interface so other implementations can be introduced as well.

Universal data storage systems can be accessed and queried too, as long as they provide HTTP-based access. This typically applies by default for the family of NoSQL databases (ElasticSearch, MongoDB, Cassandra, CouchDB etc.). In case of more traditional relational (SQL) databases which are also used as full-blown corpus search engines (e.g. The Corpus del Español (Davies, 2005) at English-Corpora.org (Davies, 2008)) the problem is not yet addressed by WaG but we are examining possible solutions. For example, in case of the PostgreSQL database, the PostgREST project (Nelson, 2017) provides a general HTTP REST API above any PosgreSQL schema.

4.6. Configuration

WaG set-up and deployment is very easy since an administrator just selects suitable tiles, data services and LRs, and finally defines a required layout.

Specifically, the application is configured via two main JSON files. One for the server properties and one for the client set-up (where all the tiles and layouts are defined). The client configuration can be further split into smaller files (general configuration, tiles, layouts, chart colors) for easier editation. WaG provides JSON schemata and validation scripts for all the configuration files which makes the process of creating the set-up more convenient and less error-prone. An example of a tile configuration can be seen in Figure 5 and an example of a layout configuration can be seen in Figure 6.


```
"cs": {
  "single": {
    "groups": [{
      "groupLabel": {
        "en-US": "Written language",
        "cs-CZ": "Psany jazyk"
      },
      "groupDescURL": {
        "en-US": "/path/to/freqProfile.en.html"
        "cs-CZ": "/path/to/freqProfile.cs.html"
      },
      "tiles": [
        {
          "tile": "CollocationsSynchronic",
          "width": 1
        },
        {
          "tile": "CollocExamplesSynchronic",
          "width": 2
        }
      ]
    }]
  }
}
```


Figure 6: A layout configuration example

5. Implementation

As already mentioned in Chapters 2. and 4.5., WaG is an end-user web application where most of the work is left up to the client-side. The reason for this lies in the asynchronous nature of the result presentation where all the tiles handle their data fetching as well as data rendering mostly[1] independently and concurrently. This makes the user interface more responsive because as soon as a tile obtains its data response, it can be immediately shown to a user while other tiles may still process their data.

The decision to put most of the logic to the client-side along with our experience with developing corpus search interface KonText and other CNC applications has led us to an idea to implement WaG as so called isomorphic (or universal) application where most of the code (in case of WaG, it is TypeScript) is shared between server and client with the server part running on *Node.JS* runtime. The isomorphic design allows us to:

- reduce duplicated code involved in data exchange between client and server (as compared to a more traditional design where similar code must be written e.g. in PHP and JavaScript for server and client respectively),
- use single data and code for handling user interface texts translations,
- render user interface components on both client and server.

Using a typed variant of JavaScript called *TypeScript* proved to be an effective solution for handling WaG codebase (currently about 32.000 LOC [2]). In our experience,

[1] Definable tile dependencies discussed in 4.4. put some limits to this as dependent tiles must wait for intermediate results produced by their "source" tiles.

[2] Lines of code.

strictly typed code makes regular changes in the application much less error prone with many errors caught during the compilation process. Also, having formally defined abstract interfaces and data types makes reasoning about the code and its understanding easier [3].
The *view* part of the application is written using *React* library which provides component-based design with clear component lifecycle and state handling.
WaG application *state management* and *event handling* is written using *Rx.JS* library which is based around the concept of *Reactive programming*. This part is loosely inspired by libraries/approaches like *Flux* and *Redux*.
The WaG implementation in general emphasizes functional programming style by prefering pure functions, data immutability, function composition and careful treatment of side-effects produced by actions within the application.

5.1. Extensibility

WaG functionality can be extended mainly by:

- creating new tile types,
- developing new data services.

The process of making custom tiles and data services in WaG has been designed in a way allowing developers to avoid interfering with the core source code.
To create a new tile, a developer makes a directory within *src/js/tiles/custom* directory with an entry module *index.ts*. The entry module is expected to export a tile type identifier constant TILE_TYPE and a factory function TileFactory creating a tile instance. The tile instance must implement a single interface ITileProvider which mainly addresses testing for tile's different capabilities and properties. The tile logic itself is handled by a *model* object which must respond at least to a RequestQueryResponse action message and produce at least a TileDataLoaded action message notifying WaG that the tile is ready to render its data.
Alternative data services for individual tile types can be written by implementing respective interfaces defined in *src/js/common/api/abstract*. Custom tiles with data services not described by those interfaces can include any needed data service code within their "home" directories.

5.2. Performance

Given by its purpose and architecture, WaG client-side application creates many individual requests to all the configured LRs which may put a significant short-term load on LR servers (Figure 7). This can be addressed by:

- *Client-side caching* (part of WaG) where web-browser application stores few recent results to handle situations like clicking the "back" button without involving additional network and server load.
- Accessing LR services via a *caching web proxy server* which sits between WaG and those services and which can reuse responses from already dispatched requests.

[3]There is no clear consensus on this among software developers which means that we present our individual experience here rather than a general statement.

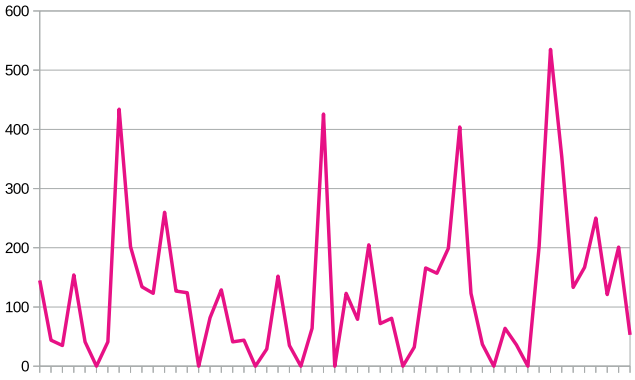

Figure 7: A typical measured load (in num. of requests) on a WaG data service backend with large, narrow peaks (time step: 15 minutes)

5.3. Development process

The WaG project is open-source (Apache 2 license) with development coordinated via a GitHub repository at `https://github.com/czcorpus/wdglance`. A production version run by the CNC has been launched in September 2019 at `https://www.korpus.cz/wag/` with currently about two thousand user queries per week. We welcome any comments, bug reports and pull requests.

6. Outlook

The development takes place in two main areas. The first and more practical one is aimed at creation of new tiles and LR APIs to offer additional datasets and functionality. The second and more abstract area concerns the core of WaG where we will work towards a better formalization of the representation of searched words and their relationship for easier integrability with other services and LRs.

7. Conclusion

WaG is a universal language resource integration platform equipped with many ready-to-be-used modules (tiles) covering typical corpus-related research topics. Its bulding kit-like architecture and some of the recently created tiles open it also for use in the broader area of digital humanities. Although its primary purpose is now to serve as a part of the CNC web portal, its design allows easy deployment and customization by other LR providers and research projects. WaG is developed with both newcomers and advanced users in mind. The newcomers can easily see the power of the data, while the advanced users appreciate it as an overview of typical word's behavior and a portal connecting more advanced and specialized tools. The general message is: "wherever you go, just start with WaG that may show you something interesting you didn't know about." We believe this could greatly improve not only the usability of many infrastructures, but also attract potential users and bring them closer to the language tools and technologies.

8. Acknowledgements

This paper resulted from the implementation of the Czech National Corpus project (LM2018137) funded by the Ministry of Education, Youth and Sports of the Czech Republic

within the framework of Large Research, Development and Innovation Infrastructures.
The author would also like to thank Michal Křen for valuable comments on this article.

9. Bibliographical References

Cvrček, V. and Vondřička, P. (2011). Výzkum variability v korpusech češtiny. *František Čermák (ed.): Korpusová lingvistika Praha 2011. 2. Výzkum a výstavba korpusů.*, page 184–195.

Davies, M. (2005). The advantage of using relational databases for large corpora: Speed, advanced queries, and unlimited annotation. *International Journal of Corpus Linguistics*.

Davies, M. (2008). English-corpora.org. `http://english-corpora.org`. Accessed: 2019-11-20.

Kilgarriff, A., Baisa, V., Bušta, J., Jakubíček, M., Kovář, V., Michelfeit, J., Rychlý, P., and Suchomel, V. (2014). The sketch engine: Ten years on. *Lexicography*.

Klein, W. and Geyken, A. (2010). Digitales wörterbuch der deutschen sprache. das wortauskunftssystem zur deutschen sprache in geschichte und gegenwart. *hrsg. v. d. Berlin-Brandenburgischen Akademie der Wissenschaften*.

Machálek, T. (in press). Kontext: Advanced and flexible corpus query interface. In Nicoletta Calzolari, et al., editors, *Proceedings of the Twelfth International Conference on Language Resources and Evaluation (LREC 2020)*. European Language Resources Association (ELRA).

Nelson, J. (2017). Postgrest. `http://postgrest.org`. Accessed: 2019-11-20.

Škrabal, M. and Vavřín, M. (2017). The translation equivalents database (treq) as a lexicographer's aid. In Iztok Kosem, editor, *Electronic lexicography in the 21st century. Proceedings of eLex 2017 conference*, pages 124–137. Lexical Computing.