Bringing DFG Project Nr. 322463161 to the Web
A Documentation

Eduard S. Lukasiewicz

under the supervision of

apl. Prof. Dr. Sascha Gaglia

September 30, 2020

Contents

Attention! This document is being constantly updated, so it is possible that the reader will not find exactly the same content each time it is consulted. Old versions of this document are stored in an online repository accessible via the following link: |—ADDLINK—|.

0 General Information about the Project

This is a documentation of the creation and deployment of a database created as part of the project Temporal analysis and modelling of the paradigmatic extension of French and Italian verbal roots. Said project was made feasible by a grant of the DFG (i.e. German Research Foundation) to apl. Prof. Dr. Sascha Gaglia, who led the project from start to finish. Acknowledged be the contribution of the following individuals to the project (the order is chronological and thus does not imply anything with respect to the magnitude of the individual's contribution):

Manuel Möll Pascal Hornbergs

Noel Seger Eduard S. Lukasiewicz

Apart from a set of publications authored by apl. Prof. Dr. Sascha Gaglia, the project involved the construction of a database of diachronic morphological verb forms of Old French and Old Italian. The data come from Frantext[1], the NCA[2] and OVI[3], which are also duly cited at the end of this section.The main goal of this file is to document the final phases of this process and to keep track of new developments in the corpus or in the interface. Consequently, we expect it to be updated regularly in the foreseeable future. Before starting to work with the tools we created, the user should make sure to be up-to-date with the newest developments in the project. The most important update will be exiting beta, which we will probably announce by the end of 2020. We therefore suggest any interested user to watch our GitHub repository: |–LINK–|. There is however no obligation on our side to do maintenance on the code or improve on it and we may drop any further development whenever we deem it necessary.

1 Introduction

The final digital output of the project consists of the following elements:

a. **FRITAV:** This is the actual corpus. We opted for PostgreSQL as its database management system as it is free and open source software (henceforth FOSS) and also because of its excellent interfacing possibilities. The version we used is 12.4 and during development we worked with pgAdmin 4 v4.25 and v4.26 for data clean-up.

b. **RhoSquared:** This is the interface to the corpus. At the moment of writing (30.09.2020), it consists solely of a command line script written in Python 3.8.2 which accepts simple query strings and outputs the corpus data in a browser window (the internal workings of RhoSquared are discussed in Section 2). It is hosted on GitHub[4] and is currently distributed under the MIT license, meaning that it is also FOSS. The script is in its beta version and still requires massive refactoring. A personal (albeit much far away) end goal for RhoSquared is an extensible open-source interface for linguistic databases.

[1] FI/FMF = *Base textuell FRANTEXT*, ATILF-CNRS & Université de Lorraine.

[2] NCA = Stein, Achim et al. (2006): *Nouveau Corpus d'Amsterdam. Corpus informatique de textes littéraires d'ancien français (ca 1150-1350)*, établi par Anthonij Dees (Amsterdam 1987), remanié par Achim Stein, Pierre Kunstmann et Martin-D. GleSSgen. Stuttgart: Institut für Linguistik/Romanistik, version 3.

[3] OVI = Larson, P. & E. Artale (2005). *Corpus OVI dell'italiano antico.* Firenze: CNR/Instituto Opera del Vocabolario Italiano.

[4] https://github.com/fritav/rhosqrd

2 RhoSquared

2.1 Preliminaries

The name **RhoSquared** derives from the code name I gave to our project when I joined in, i.e. *rhemarhizai*, which is Ancient Greek for 'verbal roots'. Going a little bit deeper than above, RhoSquared is a lightweight command line-browser-based GUI hybrid program written in Python communicating with the FRITAV server with the help of SQLAlchemy 1.3.19. Let me unwrap this:

1. **command line-GUI hybrid:** A *command line application* is designed for being used in a terminal, whereas a web-based *graphical user interface* (GUI) allows for communication with the code via a window in virtually any common browser. I say *hybrid* as it is a command line tool which uses a simple browser-based GUI (a.k.a. dashboard) to visualize query results. The tool we are using to this end is tabloo by Fabian Keller. A full GUI version is planned for the future.
2. **Python:** Python is a highly flexible scripting language with a relatively gentle learning curve. It is widely used both in industry and academy and remains one of the most popular language. The version RhoSquared is written in is 3.8.2.
3. **SQLAlchemy:** In order to work with a database you need to be able to communicate with it, and this is what SQLAlchemy does. It allows for a smooth integration of database functionalities into Python. In RhoSquared it plays a small role, albeit an important one.

2.2 Running the Script

In its current version (Beta), the best way to run RhoSquared is in a terminal (for the sake of conciseness I refer to Command Prompt and any Bash shell as 'terminal' whenever it does not make a difference with respect to the command input)[5], as cross-platform compatibility is a problem yet to be addressed.[6] In the foreseeable future it will be deployed as a Docker container[7], allowing for a more user-friendly installation.

The following walk-through considers Windows (Windows 10 - 64 bit), Mac (Mac OS 10.14.6) and Linux (Ubuntu 18.04).

Before doing anything, please check your Python version by entering the following command in a terminal:

```
▷ python --version
```

RhoSquared was developed in a Python 3.8.2 environment. As there might discrepancies between any Python 2.x version (which possibly came pre-installed) and Python 3, I suggest to

[5] If you do not know how to open a terminal window on your machine (nor what is meant by 'terminal window') please look it up, as a good understanding of how a terminal works is crucial for a smooth interaction with a command-line script such as RhoSquared.

[6] The script has been developed and tested on Windows 10 - 64 bit.

[7] I refer the curious reader to https://www.docker.com/resources/what-container.

install the latter before trying to run the script. In order to install it on Windows please go to https://www.python.org/downloads/ and download the latest release. If you are working on a Unix machine, please consult this guide for Mac and this one for Linux.

Next, let us install all the dependencies. These are saved in the file `requirements.txt`. Before running `pip` on it (we will come to it shortly), be sure that you are in a virtual environment. Creating a virtual environment ensures that the directory is isolated from the rest of the Python installation, thus keeping your base environment safe from version clashes and the like. If you do not know how to setup a virtual environment please consult https://docs.python.org/3/tutorial/venv.html[8]. Once you have created a virtual environment and after activating it (see link in Footnote 8), navigate to the script folder, open a terminal there and enter:

```
▷ pip install -r requirements.txt
```

Now you are setup to run `rhosqrd.py` (while remaining in the same virtual environment, of course!). When you run `rhosqrd.py`, a prompt will open (see Figure 1). Via that prompt you can a) initialize a new query environment (`init qenv`), b) ask for help (`help`) or c) exit (`exit`). More features are planned for future releases, but for now this is all it can interpret. The main design principles we have followed while developing RhoSquared are clarity and ease of use, thus minimizing the time spent learning how to use it, and this is reflected in its minimalism.

```
+++Welcome to RhoSquared!+++

This is a free open source command line database viewer
optimized for semistructured linguistic corpora.
RhoSquared has been developed as an interface to a French-Italian
morphological corpus which was built thanks to the DFG-funded
project 'Temporal analysis and modelling of the paradigmatic
extension of French and Italian verbal roots' by S. Gaglia.
It comprises both data from Old French and Old Italian and the
datasets are connected following a simple relation algebra.

To initialize a new query environment, type 'init qenv' below.
To display useful tips, type 'help'. To exit, type 'exit'.
>>>
```

Figure 1: The initial prompt of RhoSquared

By typing `init qenv` in the initial prompt a new query environment is initialized. This is where all the queries are processed before being sent to another module for visualization in a browser window. Upon initializing a new query environment the user is prompted to submit a query in the form of a string, as shown in Figure 2.

[8] Attention! If you do not have already, you first need to install it.

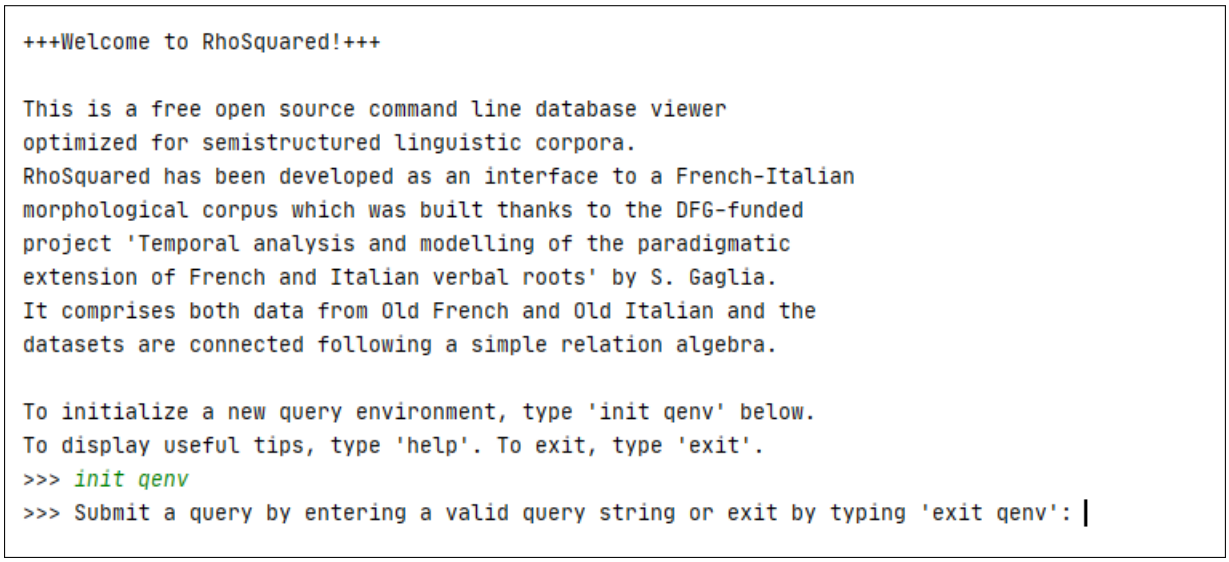


Figure 2: Initializing a new query environment.

The language of query strings, also called query language, is quite rudimentary, but using it requires knowing the corpus tags' format. This is given in Table 1 on the next page with a description for each tag. Now we can define the query language accepted by the query environment of RhoSquared. The set of well-formed (or valid) query strings is defined recursively:

1. Given a tag X and any UTF-8 string a containing only letters, numbers, commas or round/square brackets, $X = a$ is a *query atom*.
2. Given a set of query atoms $\mathbf{X} = \{(X_i, a_i) : X_i = a_i\}$, any combination of the elements of $\mathbf{X}$ by means of "&" is a *query string*.

For example, the string

▷ `writers_dialect=champenois`

will return all entries whose writer's dialect has been recognized as Champenois, whereas

▷ `writers_dialect=champenois & m_phenom=Diphthong`

will return the intersection of the set defined by the query string above with that of the entries whose identified morphological phenomenon is diphthongization. The format for temporal query atoms (i.e. query atoms $X = a$ where X is a temporal tag, referring here to `comp_dates` and `manuscr_dates`) is somewhat idiosyncratic, in that e.g. in order to query for those entries whose date(s) of composition fall(s) between 1300 and 1350, one needs to type

▷ `comp_dates=(1300, 1350)`

The limits are set within round brackets and they are *always* exclusive. For inclusive limits type

▷ `comp_dates=[1300, 1350]`

instead. The brackets can be mixed so as to represent clopen intervals.

`lang`	**Language:** Has only two values, `French` and `Italian`.
`lemma_mod`	**Modern Lemma:** The item's corresponding lemma in the modern languages.
`lemma_nca`	**NCA Lemma:** The item's lemma as found in the NCA.
`verb_form_dia`	**Diachronic Verb Form:** The actual item in the respective corpora.
`verb_form_mod`	**Modern Verb Form:** The item's corresponding form in the modern languages.
`stem`	**Stem:** Self-explanatory.
`pos_m_features`	**Position of Morphological Features:** The item's formal morphological aspects.
`pos_m_features_alt`	**Alternative Position of Morphological Features:** An alternative to the above.
`orthogr_con`	**Orthographic Context:** The graphemic context of the item.
`m_phenom`	**Morphological Phenomenon:** The morphological phenomenon affecting the item.
`db_hit`	**Database Hit:** The encoding of the query result in the corresponding databases.
`comp_dates`	**Date(s) of Composition:** Self-explanatory. The bracketed plural marked in date(s) hints at the fact that some text cannot be dated precisely.
`comp_loc`	**Locus of Composition:** The area or place where the item's text originated.
`manuscr_dates`	**Date(s) of Manuscript Composition:** The date(s) of the composition of the manuscript in which the text is preserved. Not available for Italian data.
`manuscr_loc`	**Locus of Manuscript Composition:** Parallel to above.
`writers_dialect`	**Writer's Dialect:** The dialect of the item's text. Not available for Italian data.
`reg_codes_dees`	**Regional Codes (Dees):** The item's encoding according to Dees.
`tok_sentence`	**Token Sentence:** The item's syntactic context.
`verses`	**Verses:** Yes/No value depending on whether the item's text is in verse form.
`genres`	**Literary Genre:** Self-explanatory.
`comment`	**Comment:** Self-explanatory.
`contrib`	**Contributor:** A code corresponding to the person who added the item's entry to the database.

Table 1: The corpus tags and their meanings.

To return all entries whose date(s) of composition fall(s) before or after any given date, use

▷ `comp_dates=(0, 1350)` and

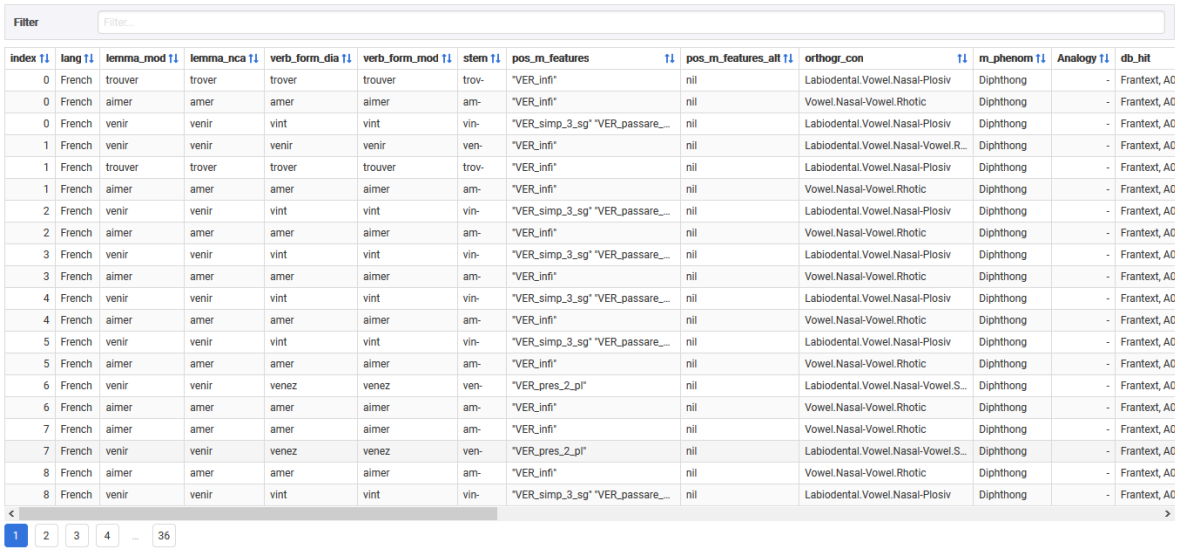
▷ `comp_dates=(1300, 0)`

respectively, while still respecting the conventions mentioned above. Warning! In the current version every query string may contain one and only one temporal query atom. This will be changed before exiting beta testing.

When you submit a query such as

▷ `m_phenom=Diphthong & comp_dates=(0, 1200)`

a window will open in the user's default browser and it will look like Figure 3 below.



Filter

index	lang	lemma_mod	lemma_nca	verb_form_dia	verb_form_mod	stem	pos_m_features	pos_m_features_alt	orthogr_con	m_phenom	Analogy	db_hit
0	French	trouver	trover	trover	trouver	trov-	"VER_infi"	nil	Labiodental.Vowel.Nasal-Plosiv	Diphthong	-	Frantext, AC
0	French	aimer	amer	amer	aimer	am-	"VER_infi"	nil	Vowel.Nasal-Vowel.Rhotic	Diphthong	-	Frantext, AC
0	French	venir	venir	vint	vint	vin-	"VER_simp_3_sg" "VER_passare_...	nil	Labiodental.Vowel.Nasal-Plosiv	Diphthong	-	Frantext, AC
1	French	venir	venir	venir	venir	ven-	"VER_infi"	nil	Labiodental.Vowel.Nasal-Vowel.R...	Diphthong	-	Frantext, AC
1	French	trouver	trover	trover	trouver	trov-	"VER_infi"	nil	Labiodental.Vowel.Nasal-Plosiv	Diphthong	-	Frantext, AC
1	French	aimer	amer	amer	aimer	am-	"VER_infi"	nil	Vowel.Nasal-Vowel.Rhotic	Diphthong	-	Frantext, AC
2	French	venir	venir	vint	vint	vin-	"VER_simp_3_sg" "VER_passare_...	nil	Labiodental.Vowel.Nasal-Plosiv	Diphthong	-	Frantext, AC
2	French	aimer	amer	amer	aimer	am-	"VER_infi"	nil	Vowel.Nasal-Vowel.Rhotic	Diphthong	-	Frantext, AC
3	French	venir	venir	vint	vint	vin-	"VER_simp_3_sg" "VER_passare_...	nil	Labiodental.Vowel.Nasal-Plosiv	Diphthong	-	Frantext, AC
3	French	aimer	amer	amer	aimer	am-	"VER_infi"	nil	Vowel.Nasal-Vowel.Rhotic	Diphthong	-	Frantext, AC
4	French	venir	venir	vint	vint	vin-	"VER_simp_3_sg" "VER_passare_...	nil	Labiodental.Vowel.Nasal-Plosiv	Diphthong	-	Frantext, AC
4	French	aimer	amer	amer	aimer	am-	"VER_infi"	nil	Vowel.Nasal-Vowel.Rhotic	Diphthong	-	Frantext, AC
5	French	venir	venir	vint	vint	vin-	"VER_simp_3_sg" "VER_passare_...	nil	Labiodental.Vowel.Nasal-Plosiv	Diphthong	-	Frantext, AC
5	French	aimer	amer	amer	aimer	am-	"VER_infi"	nil	Vowel.Nasal-Vowel.Rhotic	Diphthong	-	Frantext, AC
6	French	venir	venir	venez	venez	ven-	"VER_pres_2_pl"	nil	Labiodental.Vowel.Nasal-Vowel.S...	Diphthong	-	Frantext, AC
6	French	aimer	amer	amer	aimer	am-	"VER_infi"	nil	Vowel.Nasal-Vowel.Rhotic	Diphthong	-	Frantext, AC
7	French	aimer	amer	amer	aimer	am-	"VER_infi"	nil	Vowel.Nasal-Vowel.Rhotic	Diphthong	-	Frantext, AC
7	French	venir	venir	venez	venez	ven-	"VER_pres_2_pl"	nil	Labiodental.Vowel.Nasal-Vowel.S...	Diphthong	-	Frantext, AC
8	French	aimer	amer	amer	aimer	am-	"VER_infi"	nil	Vowel.Nasal-Vowel.Rhotic	Diphthong	-	Frantext, AC
8	French	venir	venir	vint	vint	vin-	"VER_simp_3_sg" "VER_passare_...	nil	Labiodental.Vowel.Nasal-Plosiv	Diphthong	-	Frantext, AC

1 2 3 4 ... 36

Figure 3: The query results visualized in tabloo.

The window will remain open and you will be able to submit a new query, which will generate a new window, and so on until you type `exit qenv` in the query environment prompt (or force quit, although I strongly advise against it!). In fact, the windows will remain open even *after* quitting the script, although the threads on which the corresponding processes are running will close as they are instantiated as daemons. This was the easiest solution given the short development time. In a future version, threading will be factored out.

3 Known Issues

In random order:

- **Valid Query Strings:** There is no robust method for differentiating well-formed query strings from any other string. We are planning to implement a DFA to take care of that.

- **Complexity:** The script is not optimized *at all* and requires reworking, e.g. directly querying FRITAV instead of querying copies of each table. At the moment of writing (30.09.2020), this is not a problem as the database does not contain more that 10.000 verb forms, but it will be the moment we decide to grow it.
- **Temporal Expressions:** Queries which include temporal tags are quite fragile. This needs to be reworked completely.

4 Future Developments

This section is still empty and will be added in the coming weeks.