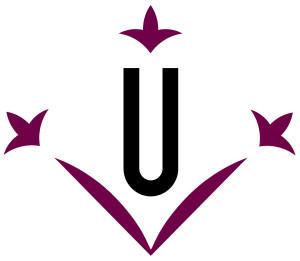
Universitat de Lleida
Escola Politècnica Superior
Departament d'Informàtica i Enginyeria Industrial



Universitat de Lleida

Generation and management of multilingual ontologies

Sergi Vila Almenara
Director: Roberto García González

Treball Final de Grau, Juny 2015

Abstract

The main objective of this project is facilitating and improving the quality of automatic translation based on online translators. The process is facilitated by providing an unified programming interface that hides the subtleties of each individual translator Web service. The API can also be reached through a command-line user interface. And the quality of the results is improved because each translation can be forwarded to the set of integrated translators, the individual results are then aggregated and then analyzed to automatically select or propose the best candidate translation, for instance favoring the one with most votes.

The result is a tool, called MultiTranslator, implemented as an independent Python package. This package contains a module (Transfusion) that offers 14 fully implemented translator services, 7 data writers to export and display the translations, a corrector to remove and fix variations of translations to obtain more uniform aggregated results. MultiTranslator is also available through three command-line applications: Transfuse, to translate terms and export translations easily, Grouper, to generate three different types of reports to simplify and share high quantity translations, and Validator, to obtain analysis about the success rate of the translated terms and the accuracy of the translators.

Multitranslator, is integrated in a larger project called NewsdeskTranslations. This project is based on the recollection of semantic data stored in a remote database using SPARQL queries according to the requests done by the users using a Django application. This information is related to the field of plant pests and other plant health threats, also including related terms like plant parts, animals or verbs. NewsdeskTranslations aim is to facilitate the enrichment of the semantic dataset to facilitate media monitoring. The tool facilitates generating and validating terms derivations (like verb forms, plurals,...), retrieving related terms and translations from external semantic datasets and also integrated MultiTranslator to translate from English to 9 languages all the selected terms.

NewsdeskTranslations integrated all these techniques to collect variations and translations of the terms in the semantic dataset. Then, after all this data is obtained and stored, the users can interactively validate the correctness of these information (called labels) using a Web-based user interface, which provides functionalities like accepting or rejecting an entire set of labels, importing and exporting sets of labels to validate them locally, a search bar, filters, sorted columns and a score system to decide more easily which labels are the best candidates for acceptance.

Acknowledgements

To my family and friends for supporting me

To Roberto and Josep Maria for starting this project and trusting me to continue it

To Juan Manuel for helping me with the documentation and the last details

To Jordi, David and Marc for being my coworkers during these last months

To Josep Maria Ribó for having been one of the best professors of the UdL

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Some decades ago the information technologies, like TV and radio, could alert about the diseases and status of the fields and the current diseases affecting these, but these news weren't related, every location was noticed about its own problems. But now, with Internet on our side, generating millions of breaking news every day around the world about anything aspect of our life, this data can be found using the correct tools to find and analyze it, but currently the problem is the complex management of this information, it is virtually untreatable.

Nowadays some organizations are taking care about the preservation of the Nature and the environment, helping the people to prevent imminent pests and diseases, one of these organizations started a new project exploiting the capacities of Medisys, an automatic system that recollects news about human, animal and plant diseases, chemical, biological, radiological and nuclear threats, and food and feed contaminations.

This project, currently named NewsdeskTranslations, was started by Roberto, my TFG's tutor, and Josep Maria, among others professors and professionals. The development of this tool will take more than two years, so my job comprises specific goals according with the temporalization. In this project was developed a Django application to obtain related terms associated with a list of pest available in a remote database accessible thought SPARQL queries. After the data collection, it is validated by the users and sent again to the remote database, the final objective of the project is the use of these labels to create word filters for an external tool, that in this case, it is used to detect emerging pests. In addition, the main page where the users request the pest information has undergone radical changes, with a complete redesign and implementing some features for a more comfortable use.

Some names of technologies used are omitted to preserve this private data, so some parts of the project are simplified.
After the investigations of the current available technologies, a Python module to perform translations was made, MultiTranslator, including useful command-line applications to translate and store terms, get the best translations based on criteria and verify the accuracy of the translators and the success rate of the translated terms.

1.1 Motivation and Objectives

The idea of using the chain of technologies present in this project, being this the intermediate gear of it for a bigger cause was one of the first motivations.
Also the Python language was a plus, during the degree some professors motivated us to use this language for its simplicity and fast development, I feel comfortable working with it, and new knowledge that I never learn before has been obtained here. Django is another technology known by me, almost the basic concepts, so I could ensure that the learning of new technologies wouldn't slow down too much the development. One of my fears was the web development, but as you will see at the end of this document, finally I have learn the necessary concepts to create a better tool.
Finally, the dedication and explanations of Roberto and Josep Maria finally convinced my to accept the project, knowing beforehand the tasks (and problems) that I faced.

Initially the objectives were defined as guidelines, the course of the project could derive in another ways depending on the current state of the art, but ever taking in account the next main objectives:

- Continue the development of the project, including new features both on the generation and in the management of labels.
- Do research about the existing translators on the web, tools to derive words and new ways to obtain information using the Semantic Web.
- Integrate the obtained tools in the project.
- Verify the correctness of these tools.
- Implement new features to supply new discovered needs during the development.

1.2 Timing

The Final Degree Project has a duration of 375 hours split in 4 months, that is, approximately 23 hours per week, so to reach these amount of hours I decided 4 hours per day is fine, the last 3 hours were distributed at the weekends and periods when I hadn't class work. Obviously, during some periods I worked more and others less, but in general, they were quite balanced.
My implication in the project officially started on February 16, some weeks before I had access to the project input materials and I began to collect and study them and decide what, when and how the identified requirements and tasks should be completed. The project finished on June 19, when this document was printed and submitted to the University, only pending it defense.

1.2.1 Distribution of the project

According to the main objectives, a general distribution of tasks and their timings was done according with the temporalization, the definitions of the tasks are enough general to decide which technologies will be used after the initial investigations.

First there is a simple planning of the first ideas of the project:

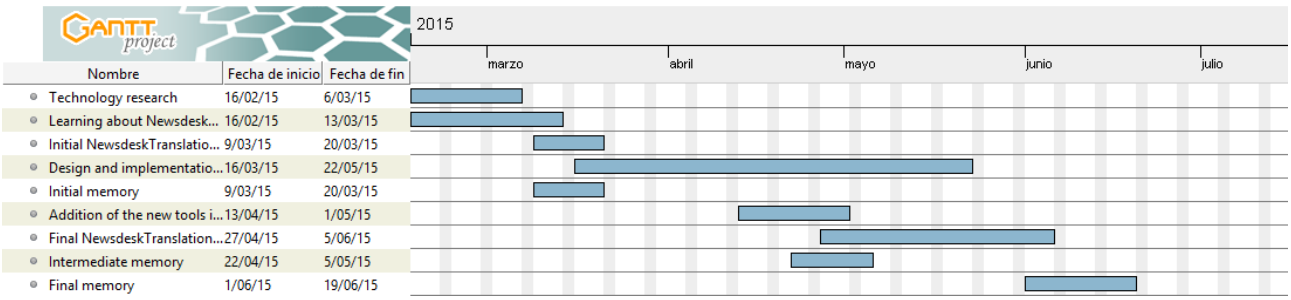


Figure 1.1: Initial Gantt chart temporalization

And then the final planning done showed in the figure 1.2

As it can be seen, the research concluded with the implementation of the MultiTranslator project, also, the design of the NewsdeskTranslations is not temporalized, but the deadlines allowed the study of the necessary technologies to complete this task.

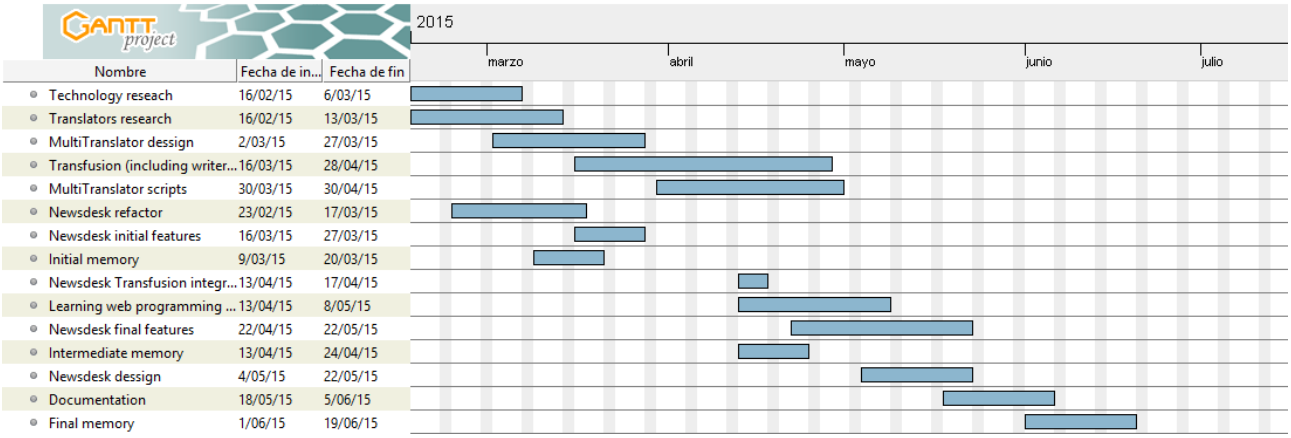


Figure 1.2: Real Gantt chart temporalization

1.2.2 Statistics

Seeing the distribution of project commits in GitHub, we can take an approach about the distribution of time in both projects. It is important to notice that GitHub only computes commits, not the amount of work behind each commit. It may occur that one feature takes one day of work and 5 commits (because the code needs a lot of improvements) while another requires a couple of days but only one commit is done, as only finished features were committed. The next images shows the implication in the project over time.

The columns are the days of the weeks starting by Sunday, the most of the days I made commits during the work days.

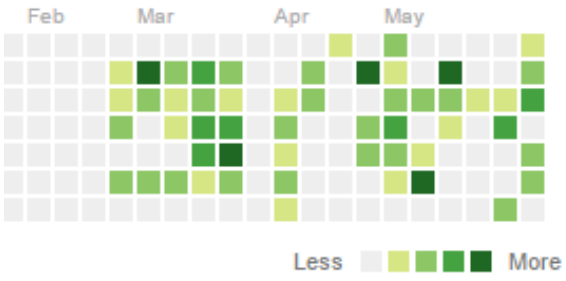


Figure 1.3: Commits history

The multitranslator project has the next commits, additions and deletions of lines:

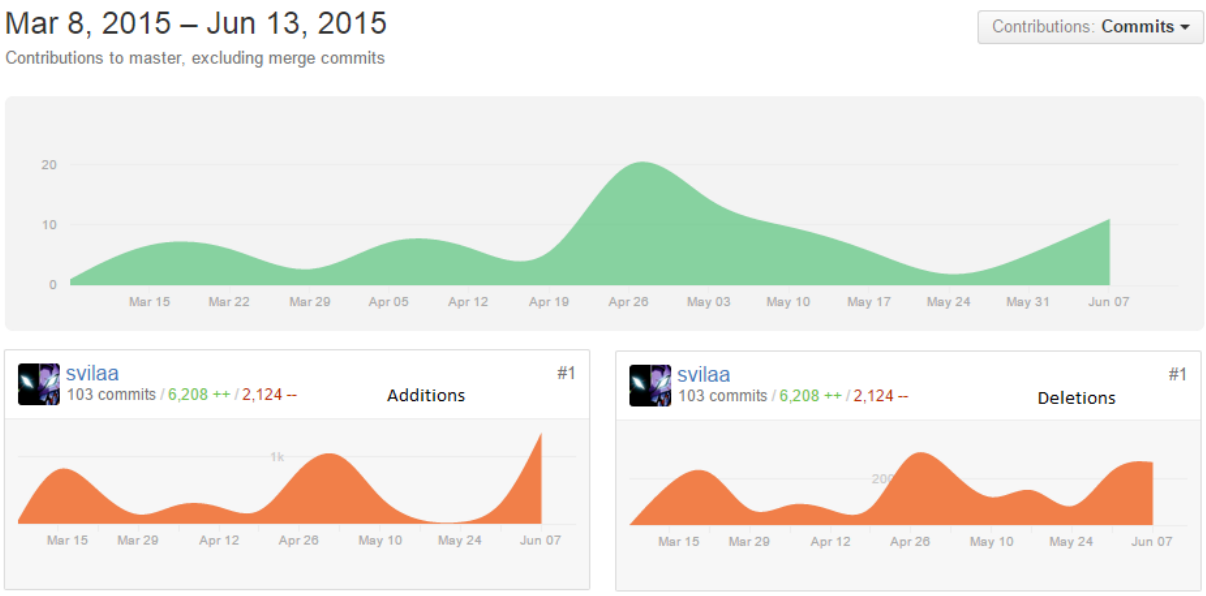


Figure 1.4: Multitranslator commits

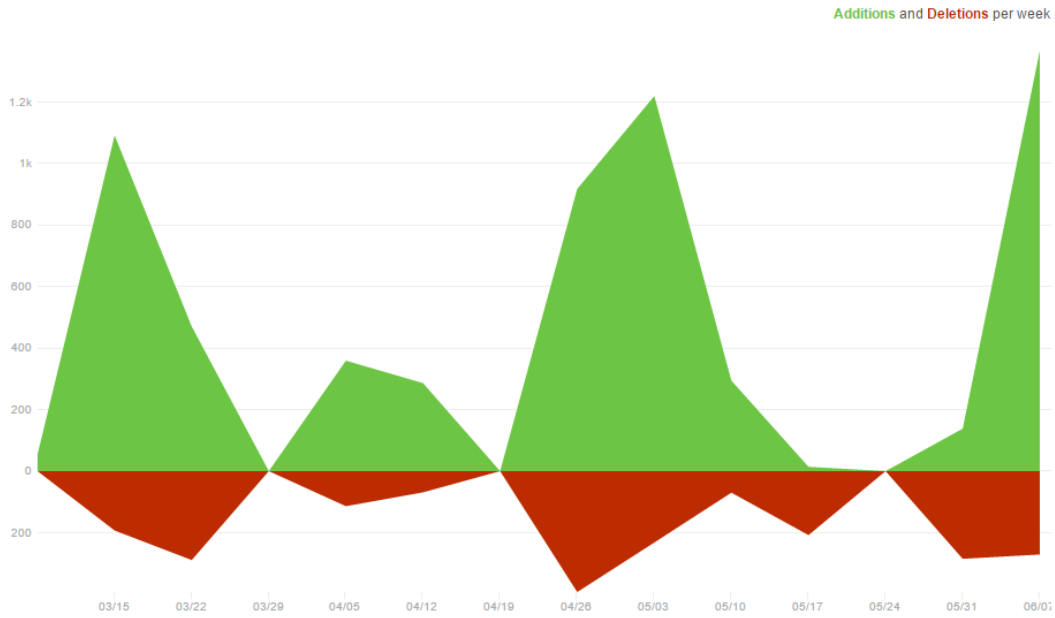


Figure 1.5: Multitranslator workweek

And the NewsdeskTranslations project:

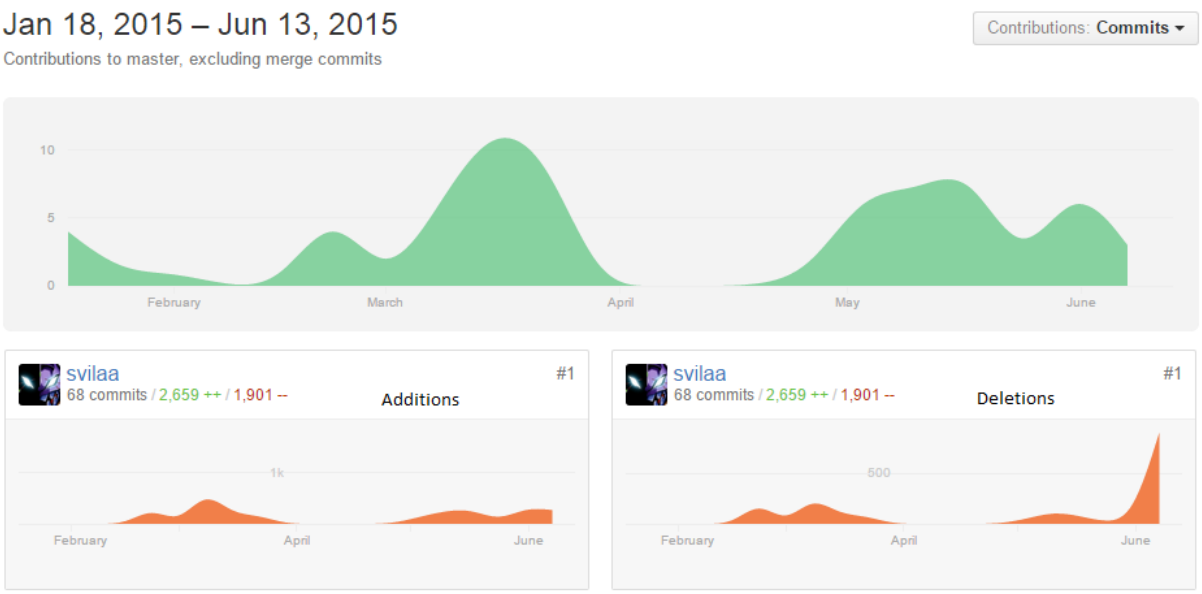


Figure 1.6: NewsdeskTranslations commits

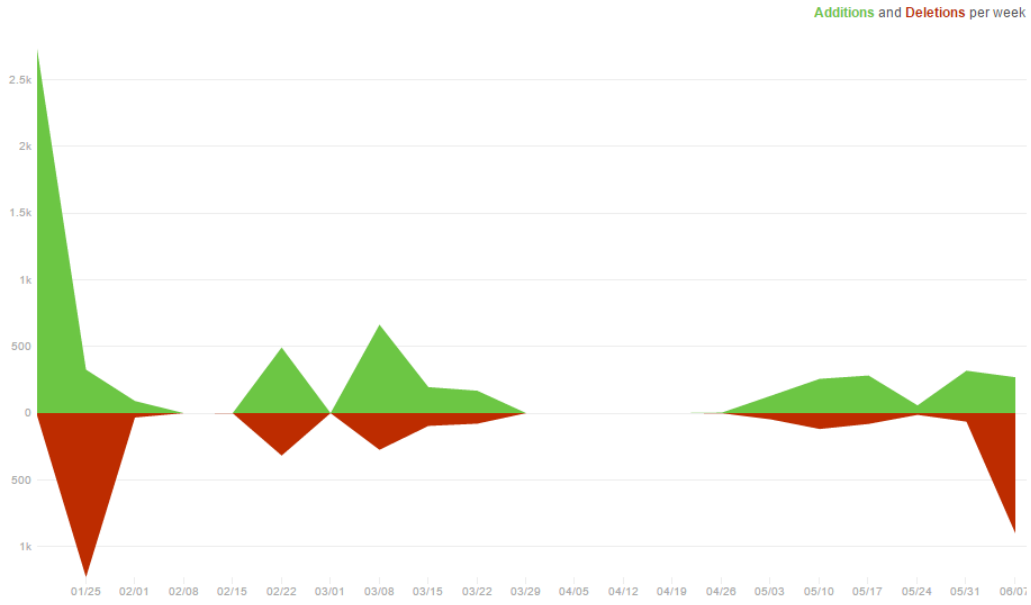


Figure 1.7: NewsdeskTranslations workweek

Notice that the NewsdeskTranslations project starts with this number of lines:

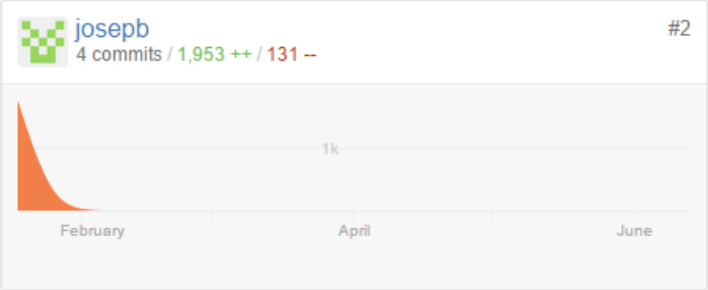


Figure 1.8: NewsdeskTranslations initial state

1.3 Costs

This project is all-based on the creation of new software, hence there isn't any cost of material, nor software licenses. But indirectly I used products and I had a workplace at the GRIHO laboratory of the Universitat de Lleida.

The tools used for developing were my personal laptop and its accessories, no additional hardware was required.

Most of the programs used are free, had free plans or a demo period, like Google Chrome, LibreOffice Calc, SourceTree, SublimeText or VirtualBox, including the PyCharm and Github student licenses. Others are not free, but I had these before the project started, like Windows 8.1 and Microsoft Word.

For now the hardware and the software are covered, no additional spends are required, but any software development has another important cost, the programmers and their salary. Looking the labor agreement at the Boletín Oficial del Estado[1] we can find the salary for a junior programmer, that is fixed in 13.634,88 euros per year and 40 hours per week, but remember the development lasted 4 months and every week I work 23 hours on average. So we need to calculate the equivalent salary for this project:

$$13634,88\ euros * \frac{4\ months}{12\ months} * \frac{23\ hours}{40\ hours} = 2613,352\ euros$$

So the final cost of the project is 2613 euros.

If this was a real budget, this cost may vary if some considerations are taken, first, if additional fee licenses are required, part of the cost could be assumed by the client, also including the amortization of

[1] `https://www.boe.es/diario_boe/txt.php?id=BOE-A-2009-5688`

the hardware, and second, the client can assume that some technologies are known by the developers, in this project most of them are known, like Python and Django, but others not, including JavaScript and web design, so these last extra technologies are not assumed by the client.

Chapter 2

State of the art

2.1 Introduction

Before the development of the applications, some investigations were done to enter to the fields of the semantic web, word derivation and translations to decide which parts of the project finally can be implemented.

2.2 Semantic web

We would think semantic web is a trendy term that is living a great expansion due to the new technologies based in the Web 3.0. But actually it was the original idea of Tim Berners-Lee, it was his first attempt to develop an interconnected network where all the data could be identified and related, which would help the machines to understand human concepts, providing a new way to manipulate the massive information available. This project failed and became the hyperlinked net of HTML pages that forms the World Wide Web, but for some years, this initiative returns, extending and enriching the metadata of the existing pages to develop new technologies to access the information contained in these.

2.2.1 Terminology

Semantic web is composed by acronyms, lots of them, that describe standards and technologies, and the combination of these the objectives of the original concept of the semantic web is reached and expanded without limitations.

URI

An Universal Resource Identifier is an URL that identifies one, and only one, resource. The fact they are URL doesn't mean that can be addressed as a normal webpage, so not all the URIs has a visual representation if the domain doesn't support it. Two or more different resources can have the same name, for example, names of people, so if we search about Tim Berners and appears two different URIs, that is because one is the creator of WWW and the other is a completely different person, hence, the URIs must be different. Some ontologies assign easy names like Tim_Berners(Professor) or Tim_Berners(Actor) to identify and distinguish the resources, but others use random combinations of characters and numbers or UUID.

RDF

The Resource Description Framework is a standardized markup language based on triplets emerged from the Meta Content Framework (MCF), a system to describe and organize web content, and the XML format. A triplet is composed of three elements, all of them URIs:

- Subject: It represents a concept for the humans, can be everything you can imagine, but only represents the idea, not a particular aspect of this.
- Property: The point of connection between the subject and the object.
- Object: Another subject if the property is a connection or a final leaf (called literal) if it is an attribute like text, numbers, coordinates, images, dates.

RDFs

With RDF we can give meaning, characterize and relate concepts, but all of these are independent between them, so a new standard extension for RDF was published by the W3C in 1998, the RDF

Schema. The main features of this standard is the possibility to extend a resource from a class, providing all the properties of the base class and creating a better grouping of concepts, also this helps at the time of defining ontologies and knowing which properties have a resource of some class, for example, if we are talking about a musician, it has information like music genre, played instrument, band, albums, singles, tours, but not only these type of properties, a musician is a person, so after some extends like Musician → Artist → Professional → Person → Human → Animal → Thing, all the properties of the superclass also are inherited (name, height, birth, artistic name, feeding type, kind, location).

RDFa

The RDF information not only needs to be created and processed following the standardized markup languages, we can exploit the possibility of "hacking" the HTML labels and add semantic data on plain HTML code of a normal webpage, and combining with the described RDFs classes, a chunk of code can be automatically processed and understood, enabling new possibilities over searches, filtering better the data and thus creating an hemeroteca of own content.
The site schema.org provides a large list of classes and properties to use inside webpages about practically all the concepts of the real world.

OWL

In the RDFs definition are introduced the ontologies, an ontology is a vocabulary over a domain that defines the entities and relationships between them, providing meaning understood by a machine.
The Web Ontology Language is a standard defined by the W3C that extends the possibilities of RDFs, if with RDFs we can say that the classes Dog and Cat are subclasses of Animal, with OWL we can add that both resources are disjoint, that is, OWL permits the addition of metaproperties on the entities and properties. These extra properties are helpful at the time of merging equivalent resources from different databases and ontologies, converging into the same resource. For example, if a database is focused on animals' scientific names an another centered on animals' locations, if both databases use different nomenclatures for the classes but we are able to found a relation between them, finally we will get an ontology with all the data.
A famous and used ontology is FOAF (Fiend of a Friend), it describe the relations of social networks.

Translations ontology

To understand, complement and relate the developed translation tool with the Semantic Web project was created an ontology about translations, modeling the classes involved in the project and their relations.

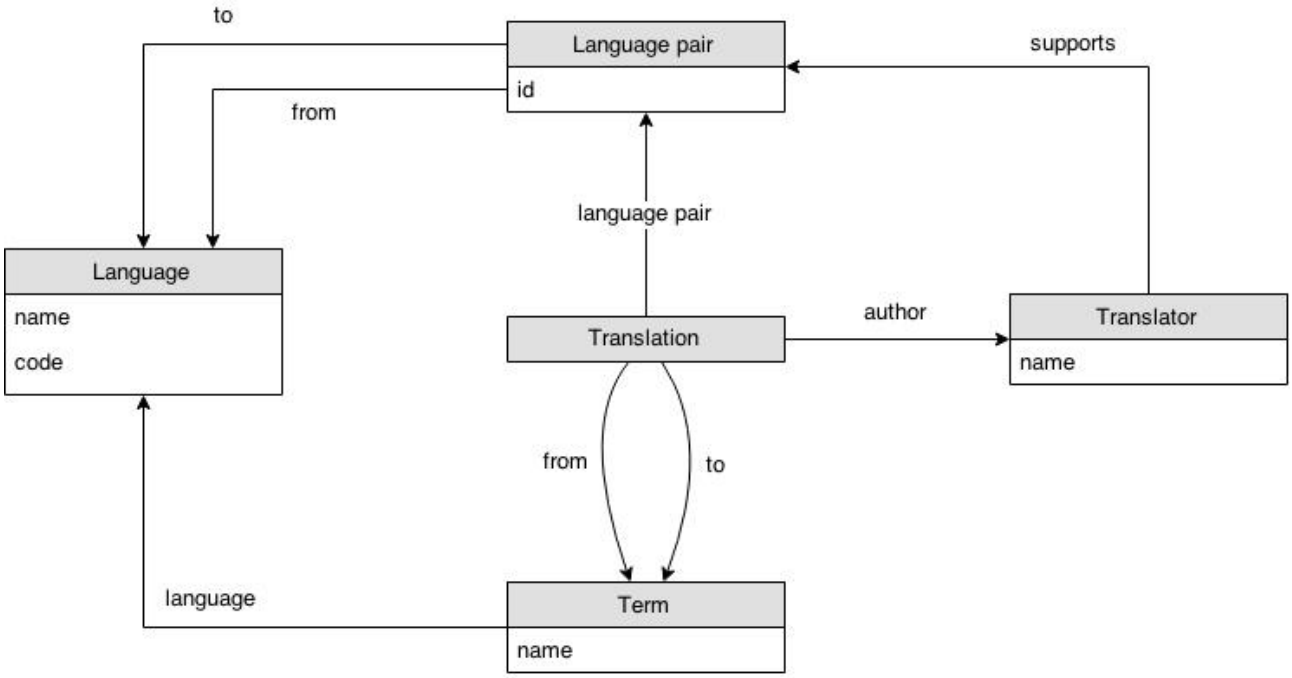


Figure 2.1: Translations ontology

- A Language has a name (English) and a code (en).
- A Language pair is a combination of two Language entities (to and from, English and Spanish), and also has an id (en-es).
- A Term is a concept of our reality, name can be a single word (hello), a combination (good bye) or a phrase (change color), in short, the text that is translated. A term is displayed in a specific Language.
- A Translator has a name (Google) to identify it and the language pairs that it supports.
- A translation is created by a Translator (the author), has a source term (from) and its translation (to), the languages of the terms encodes an equivalent language pair.

SPARQL

The SPARQL Protocol and RDF Query Language is the query language used to retrieve information from Semantic Web databases, even allowing the creation and update of content. Tim Berners-Lee

did a comparison with relational databases to understand the importance of SPARQL:

"Trying to use the Semantic Web without SPARQL is like trying to use a relational database without SQL."

Parts of a query

Prefixes

All the information which is not a literal is a resource, so all the properties, classes and other resources are URIs, in order to facilitate the comprehension of the queries and to shorten these, at the start of the query, is very helpful first define the root of the URIs, called namespaces.

Select

This is the information that will be retrieved, like an SQL query. All the variables must start with an interrogation. The action DISTINCT is recommended to avoid duplicated information. It has the next structure:

```
SELECT [DISTINCT] ?<variable1> ?<variable2> [...]
```

From

In most of the cases this is not necessary, but if we need to work with complex queries that involves specific resources or databases, a set of queries only be directed to a group of resources, and others to another, so using FROM and FROM NAMED this issue can be solved.

Where

Here go the imposed restrictions over a domain of data to find the requested information. The instructions are wrote using the triplets' syntax:

```
?<variable> <property> [literal | ?<variable> | <URI>]
```

The restrictions are wrote following the next format:

```
?object foaf:name ?name
```

For multiple restrictions (the most common case), all the instructions must end with a dot except the last. If a left variable is used more than once, a semicolon at the end of the line avoids the repetition

of this variable, only indicating the property and the other resource or literal.
Additionally the action FILTER can be used for boolean conditions about the used variables.

Options

Actions inherited from SQL like ORDER BY, LIMIT, OFFSET, UNION can be used with the same effects as this.

Endpoints

SPARQL it's only the language used for the queries, standardizing these for all the databases, but it is not enough, services called endpoints get the queries to return the results, normally every database based on an ontology has its own endpoint, because it is who has direct access to the data, that is why we don't need to indicate the used graph, because by default, the endpoint uses itself. Furthermore, some endpoints has the capability to combine other endpoints and databases to retrieve data for all the supported sites. At last, some endpoints need authentication to manage the number of queries and to avoid the abuse of the service.

2.2.2 Input of the data

There are many ways to add or enrich resources using automatic processing, interactive user interfaces and existing databases. The preferred system is the content managed by people, because one of the problems found with automatic parsing is the inconsistency of the data, that is, use of different labels for the same concept, duplicated properties with different content, lack of content for an specific scope, incompatibilities between ontologies... All these problems can be solved if the person responsible of a content also adds the appropriate tags if for example is a Wikipedia article, or checks periodically one by one the properties to ensure high-quality resources.

DBPedia

It is a project created in 2007 to extract content from Wikipedia and adapt it as linked data. Currently some important Wikipedias have its equivalent DBPedia, these are independent of each other (but relatively interconnected) with specific tags and content. Obviously, the more pages has the more content has, so in this case, the English Wikipedia/DBpedia is which has more entries.

There is an example of how work it:

Some of the content from Wikipedia is formatted using templates, there are a large number of template for general information like persons, spices, music, films and sport competitions. This is the personal information template of Tim Berners-Lee:

Figure 2.2: Tim Berners-Lee Wikipedia information

There is basic information that more or less every noted person has. This information is typed and when the collector of data gets this page, it reads the key-value data (could be RDFa) and adds triplets of these resources.

Our next task is to obtain the same resource from DBPedia, so we are going to do a query using SNORQL[1]:

```
SELECT ?name ?person
WHERE {
    ?person a <http://dbpedia.org/ontology/Person> .
    ?person foaf:name "Tim Berners-Lee"@en .
    ?person foaf:name ?name
}
```

Listing 2.1: SPARQL query

[1] http://dbpedia.org/snorql/

Notice that this query is very specific, the category Person is known, the name is exactly this and the @en suffix is declared. From my experience, this isn't a query about unknown information, so a more general (and costly) query is the next:

```
SELECT ?name ?person
WHERE {
        ?person foaf:name ?name .
        FILTER regex(?name, "Tim Berners")
}
```

Listing 2.2: SPARQL query

We search all the people whose name match Tim Berners.

GeoNames

It is a geographical database with over 9 million unique resources catalogued into 645 categories (cities, rivers, shops, buildings, geographical and so on). The resources come from the local databases of each country in the world, for example, in Spain the two organizations that provide data are the Instituto Nacional de Estadística and the Instituto Geográfico Nacional. Users can edit and complement all the information using a friendly interface based on Google Maps, but the most powerful tool is the possibility to make queries using a SPARQL endpoint with its own ontology and namespaces not only about with specific fields matching, but also proximity to coordinates or delimited areas, the own site provides GeoSPARQL with examples to learn. But another excellent feature is that if a resource is linked to a Wikipedia article, this also will be linked to DBPedia, so this is a good example how different databases finally will be connected from a common URL.

MusicBrainz

Probably the biggest database about music artists and their songs. It is maintained by the non-profit organization MetaBrainz. Everyone can add or edit data following strict guidelines that finally is automatically dumped as RDF. An additional service based on fingerprints allows the possibility to compare songs using unique identifiers, the current used technologies are MusicDNS and Chromaprint.

Government open data

Since the last years the transparency of the public administrations have offered to the citizens data about their country. Most of the data are statistical tables about a specific scope, like the numbers of births and deaths along the years, population of the cities and the economic sector rates for each territory, but also demographic, social and economic information like the streets and public buildings (hospitals, schools, police stations) and services (parking lots, taxis, sports facilities) of the cities and the management of public money.
Normally every city council publish its own data and other initiatives collect it. The transparency site of Gijón[2] is a good example of a city council compromised with open data, with updated documents, many formats and also provides an SPARQL endpoint for querying.
A problem detected about this type of information is the facility to being deprecated after few months.

2.2.3 Output formats

When a SPARQL query is made, the server and the user must agree with a specific format for the future modeling and manipulation of the data, it can be RDF/XML or equivalent, but the most common formats are the next.

HTML formatting

Normally SPARQL endpoints or semantic databases have the capacity to show the linked information of a resource as a normal page only reaching the first related data. It is useful to see the fields that have a resource, to jump to other linked resources or properties, or to obtain the same resource in other semantic databases. A big deal of semantic web is the high number of properties, so when we make queries, if we want a nice performance and filtering, we must provide specific characteristics to find the requested resources. The next example shows the first ten rivers in United States starting with A:

[2] `https://transparencia.gijon.es/`

river	name
:American_River	"American River"@en
:Ammonoosuc_River	"Ammonoosuc River"@en
:Alhambra_Creek	"Alhambra Creek"@en
:Austin_Creek	"Austin Creek"@en
:Ashuelot_River	"Ashuelot River"@en
:Anacostia_River	"Anacostia River"@en
:Ahnapee_River	"Ahnapee River"@en
:Apple_Creek_(stream),_Missouri	"Apple Creek (stream), Missouri"@en
:Arkansas_River	"Arkansas River"@en
:Arroyo_Seco_Creek	"Arroyo Seco Creek"@en

Figure 2.3: HTML SPARQL output

XML

Called eXtensible Markup Language, it is a markup (that is, based on labels) language for the creation of data structures similarly to the HTML's labels. One of its essential features is that is very readable by humans due to the tree structure. Some web services like feed readers and applications that work with graphs use XML to store the data because it really be easy to treat tree structures.

```
<?xml version="1.0" encoding="UTF-8"?>
<user>
  <id>1835</id>
  <name>James</name>
  <surname>Cook</surname>
  <birth>
    <day>6</day>
    <month>3</month>
    <year>1980</year>
  </birth>
  <location>
    <city>Lleida</city>
    <region>Catalonia</region>
    <country>Spain</country>
  </location>
  <friends>
    <id>7636</id>
```

```
    <id>1284</id>
    <id>5334</id>
  </friends>
</user>
```

The first line is recommended to avoid problems about the version and the encoding. Currently there is only the initial version 1.0, so there aren't doubts about it, but the character codification is important if a service work with different languages.

JSON

The JavaScript Object Notation, as the name suggests, has the syntax of a Javascript object, that is, a tree of key-value pairs where the keys are plain text and the values can be numbers, text, boolean, arrays or another JSON objects (subtrees). The Python representation for dictionaries is equivalent to JSON, so it is a good and fast system to serialize in both sides. The codification can't be declared like XML, the service must provide a field with this data. The example is the same as the XML to show the equivalences between them.

```
{
"id": 1835,
"name": "James",
"surname": "Cook",
"birth": {
"day": 6,
"month": 3,
"year": 1980
},
"location": {
  "city": "Lleida",
  "region": "Catalonia",
  "country": "Spain"
},
"friends":
```

```
[
  {"id": 7636},
  {"id": 1284},
  {"id": 5334}
]
}
```

There exist sites to convert XML to JSON and reverse like CodeBeauty[3] and FreeFormatter[4].

CSV

The Comma Separated Values format offers a spreadsheet formatting where every text between commas fills one cell, maintaining the original structure of the plain text. This format has some limitations with texts that contains commas (phrases, directions, list of items), so there are alternative character separators to extend its capabilities like tabulations (TSV) and semicolons (the most common). Typically these type of files are imported with spreadsheet editors like Open/LibreOffice Calc and Microsoft Excel, where are configured the encoding and the character separator.

The next example shows how is encoded a receipt:

```
Product,Name,N°,Price
V-001,Potatoes,4,1.15
V-050,Salad,1,0.85
G-361,Olives,4,0.35
S-388,Bread,2,0.35
L-117,Water,3,0.45
M-343,Cereals,4,2.3
```

Notice that the decimal separator is a dot, if we use the comma, the decimal part will be next to the integer part, that is tabulations or semicolons are better and avoid problems, also can be used simple or double quotes to group text that contains commas, but if there are quotes inside the problem continues being the same.

And this is how is displayed in Excel:

[3] http://codebeautify.org/jsonviewer
[4] http://www.freeformatter.com/xml-to-json-converter.html

	A	B	C	D
1	Product	Name	Nº	Price
2	V-001	Potatoes	4	1.15
3	V-050	Salad	1	0.85
4	G-361	Olives	4	0.35
5	S-388	Bread	2	0.35
6	L-117	Water	3	0.45
7	M-343	Cereals	4	2.3

Figure 2.4: CSV example

2.2.4 Synonyms and translations

From DBPedia we have the capacity to obtain certified synonyms and translations using some properties, not all the resources have all of this, but if it has almost one, we will receive a valid term.

- rdfs:label
- foaf:name
- dbpprop:wikt
- http://es.dbpedia.org/property/name
- `prop-<language code>:wiktionary`

As a normal user, when you want to consult an article from Wikipedia in another language, you change it using the list of languages in the margin of the page, obtaining the same resource from another Wikipedia version.

The main idea is the same, using the English DBPedia to 'jump' to the other DBPedias in other languages. With the property owl:sameAs we obtain the list of equivalent terms, so we need to explore these sites and try to extract the values of the commented properties.

After some investigations and evaluating the time to found an heuristic or decision system to obtain valid these synonyms and translations finally this idea was discarded, the range of properties to obtain the correct translations vary a lot depending on the term, obtaining unclassified data in the most of the cases, and even totally incorrect and misplaced information. Also, the current disaggregation of semantic databases complicates the matter, for each one must be implemented a specific action plan studying their own particular properties.

The main problems to solve would be:

1. Find the terms in the semantic web database and certify really this is the wanted data.

2. Use the available properties of the found resources to explore related terms and translations.

An easy and fast solution could be the indiscriminate addition of all the collected data, without taking care if it is valid or not, but it would go against the ideal to find a system to validate only the required data.
A more careful option is limiting the treated properties, using only which obtain a higher rate of success in most of the cases. For example, a combination of "sameAs" to obtain related terms with "foaf:name" to finally obtain the name.
In conclusion, the current semantic web databases have the enough issues that difficult exceedingly the recollection of the required data, without guaranties that finally they can be solved.

2.3 Natural language derivation and word identification

One of the ways to obtain information about words is the use of lexical and syntactic libraries with extend number of dictionaries, corpus and thesaurus to analyze the structure of sentences. The main objective for our goal is the derivation of the original words to obtain alternative forms of the same lexema, that is the root of a set of very related terms, for example verb derivations: jump, jumping and jumped. Another important goal is the search of synonyms to obtain additional words.
NewsdeskTranslations currently uses NLTK and WordNet to obtain the plural, the lexema and alternative forms for the verbs, so this investigation doesn't start with nothing.

2.3.1 NLTK

The Natural Language ToolKit[5] is a Python library that provides complex analysis of texts, including parsing of words, word suggestions and the most important feature, statistics about thousands of books. Using these features, new plugins can be added to take advantages of the statistics, because most of the functionalities work with these, deciding which is the best output depending of the provided word or text.

[5] `http://nltk.org/`

2.3.2 WordNet

WordNet[6]s a lexical database based on synsets, that are groups of synonyms similar than a thesaurus. The basic use of the database is as a corpus integrated as a plugin into NLTK. The specific features that provides WordNet is the filtering of the terms of a sysnet from a provided type of word, comparisons between terms, verb derivation, search of the lexema, and synonyms, hyponyms and hyperonyms.

2.3.3 Dictionaries

Some online services like WordsAPI[7], Cambridge Dictionary[8], BabelNet[9], DictionaryAPI[10] provide information about the type, genre, number of a terms, and also synonyms of it. These services returns big amount of information, including an enormous range of meanings, so this data must be filtered and compared with others to obtain the searched words. For example, the term "shoot" can be the action of throwing a projectile, but also it is a sprout of a plant, if in the definitions of the terms can found the word plant, we can ensure that it is a valid translation.

2.3.4 Implementation decision

After the research some problems were detected, the lexical libraries have an acceptable word identification as long as a phrase or a medium long text is provided, but individual words without context could be a difficult task if there aren't enough services to compare, the generation of incorrect data mixed with few valid results is not acceptable. Another problem was the limitation of the languages, English is the most extended language, therefore, the quality for other languages will be lower than this.

Thinking about the derivation of words in other languages, the unique languages required for the project which I know their grammar are English and Spanish, so not only new tools need to be studied, integrated and tested, an additional extra work must be done to know how works every language. By the accumulation of the presented problems, finally the enrichment of terms is not continued and other ways must be found.

[6] `i`
[7] `https://www.wordsapi.com/`
[8] `http://dictionary.cambridge.org/`
[9] `http://babelnet.org/`
[10] `http://www.dictionaryapi.com/`

2.4 Translators

To understand the possibilities and the current status of machine translation development a brief history about the evolution of these could be interesting.
Before the investigation about the available options to develop the required functionalities, it has been performed an initial screening about probable existing applications that would be used.
Finally, an additional research was done about similar applications that group translators and have validation features.

2.4.1 History

Humanity developed along the History large variety of languages that have been evolving some from others, differentiating or converging, and finally extinguished, either for geographic, demographic, cultural, political or social reasons.
The Rosetta Stone probably is the oldest manifestation of a document with equivalent texts where some of the language are extinct, this stone made of black basalt found in a temple of Rosetta in 1799 contains the same text, the Menfis' decree, in Greek, Egyptian hieroglyphics and Egyptian Demotic. Other similar translations were found during the next years, starting a race to reveal the ancient Egypt secrets.
The first demonstration of automatic translation arose in the s.XVII using elementary translation systems using a directly word-by-word heuristic, but due to the inexistence of the computational power of the computers in this period no good results were obtained.
After that, the efforts to develop translators had serious difficulties to evolve until the development of the first computers, with IBM leading the first experiments of statistical machine translations (SMT) in the 1980s, doing a similar task than Rosetta Stone, the comparison of the same texts in different languages, with the advantage that the texts are found in Internet, offering a massive number of these. The most recent remarkable goal was the development of Moses[11], a SMT system that currently still evolving.

[11] `http://www.statmt.org/moses/`

2.4.2 Translation tools

A basic list of tools was done to start the investigations.

These are the purposed tools:

Bilingual dictionaries

Actually, where everybody has smartphone with Internet probably is less common the use of this type of dictionaries, but some years ago the tourists go with these little books that contains the equivalent words an useful phrases between two languages. Using this type of dictionaries first can checked if a term exists (or it is common, at least), and second, if it has synonyms.

Thesaurus

A thesaurus is a list of related terms with similar meaning or over the same field. Tools that can provide synonyms will be helpful, an example is words related with breath: to speak, to scream, to cry, to shout, to whistle, to snore, to expire, to whisper, to blow, to cough, to sneeze.

Translators

Services that transforms the input text from a language to another using heuristics. The more terms can be obtained, the better, it is important to found a big number of these to compare these.

Semantic linked data

Using some SPARQL endpoints and the property SameAs, genuine equivalent terms can be obtained without dubts as long as the linkages are correct.

Dictionaries and spell checkers

Some of the translations probably don't have sense, using these tools we can ensure that the terms exists and we will add more credibility to these, the meanings can help to decide which is the most appropriate.

2.4.3 Similar applications

They have been found three applications focused on translations and its management, but anything have the exactly features developed in this project, basically they helped to discover new translators.

Lingoes

Lingoes[12] is a Windows application with dictionaries and translators compatible with more than 80 languages. This program has integration of plugins, including the capacity to detect text from the browser, word pronunciation, new glossaries and language packs.

Weblate

This Django application[13] can translate using Amagama, Google, Bing, MyMemory, Glosbe and Apertium (among others), also includes a validation tool similar to NewsdeskTranslations, but it only filters be language, all the words end to the same place, and with a high quantity of words, could be very difficult their management. Finally comment the nice design it has.

Virtaal

Virtaal[14] is a cross-platform visual application to translate texts and store these in files, also some options like autocompletion, spell checking and search are included. The main remarkable features are its user interface and its easy usage.

[12] http://www.lingoes.net/en/translator/index.html
[13] https://github.com/nijel/weblate
[14] http://docs.translatehouse.org/projects/virtaal/en/latest/index.html

Chapter 3

Development

3.1 Introduction

After the initial experiments in the NewsdeskTranslation project, the decision was to combine the use of natural language tools and semantic data with translation tools, the part where this work focuses its efforts.

This section contains the design and implementation of the MultiTranslator project and the new features added into the NewsdeskTranslations project, also explaining how they are related.

The figure 3.1 shows a general overview of the entire project and how the different parts are connected.

3.2 Multitranslator

3.2.1 Introduction

MultiTranslator is a Python package that contains a set of modules and scripts to obtain, correct, display and validate translations. The main module is Transfusion, the abstraction layer to obtain translations, it is only available in a programmable environment, so Transfuse is the command-line application that uses Transfusion and offers the simplicity of a callable script with the powerful of the module. Finally, Grouper and Validator help the user to manage and correct big amounts of translations.

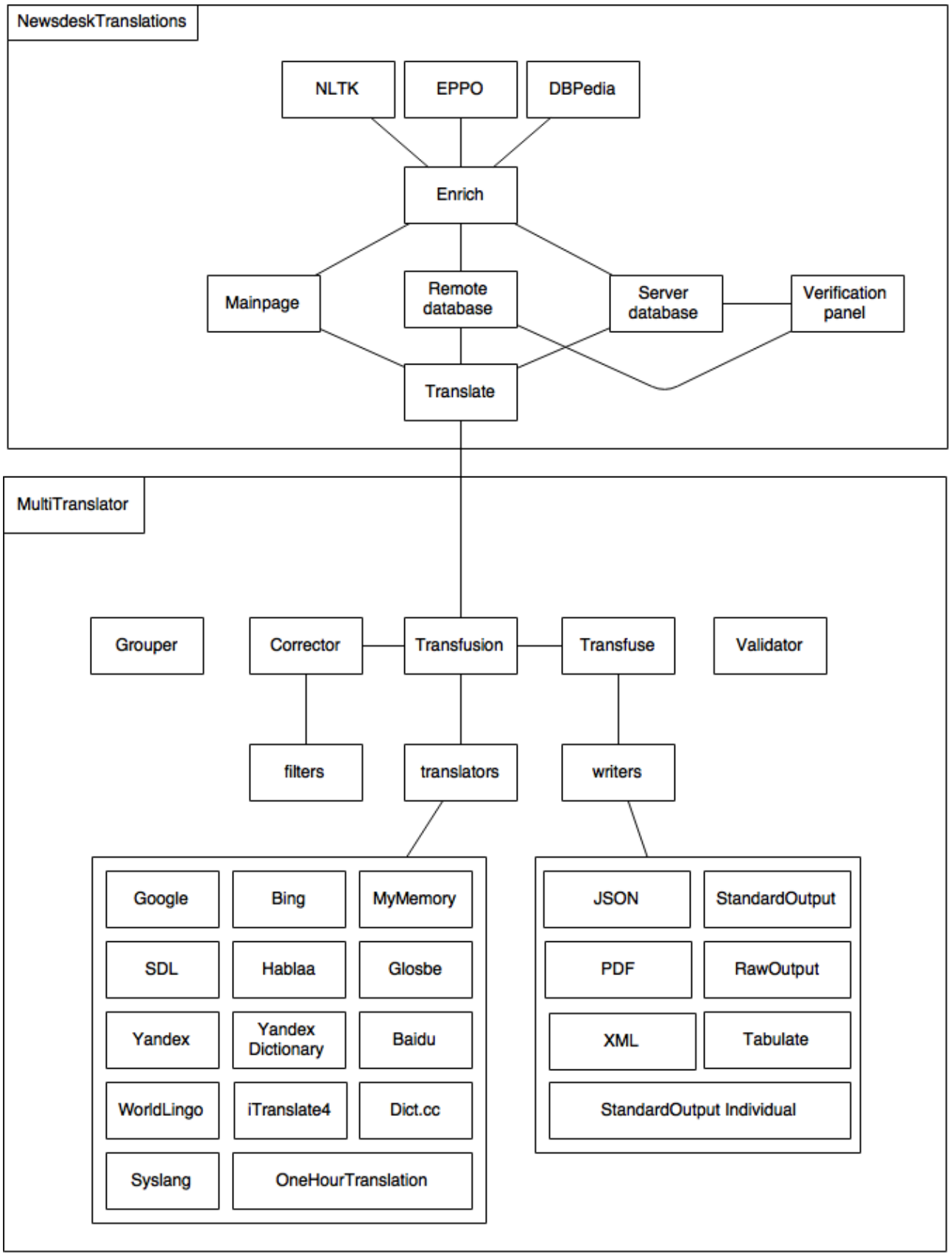


Figure 3.1: Projects overview

3.2.2 Requeriments

The functionalities required are:

- The user must be able to translate texts from a source language to a set of target languages using a default or custom configuration of translators.
- The user must be able to export the obtained translators in different formats to files or the standard output.
- The user must be able to fix translations using a configured behavior of filters execution.
- The user must be able to see the execution status of the translators during a translation process if user desires.
- The user must be able to finish any supported translation process even a translator is not working properly.
- The user must be able to use the JSON exported files to obtain spreadsheets to display the best translations and as a human validation interface.
- The user must be able to use the JSON exported files to obtain analysis about the accuracy of the translators and the success rate of the translated terms.

3.2.3 Installation Requirements

The next requirements must be accomplished to install and use the project correctly.

- A computer with a UNIX OS, Mac OS X or Windows XP or higher.
- All the online translators integrated into Multitranslator require an Internet connection.
- Python 2.7. Specifically the project was done with Python 2.7.3, to avoid incompatibilities this version is preferred. Python 3 is not compatible.
- Privileges to install Python packages.

Required libraries

The Multitranslator project depends on third-party libraries created by the community, some of them are easy to install with tools like "easy_install" or "pip", but others require platform specific components or additional libraries for the specific hardware or OS. In this cases, specific installation guidelines for some of these libraries are provided. In any case, it is highly recommended the use

of a Python virtual environment to avoid conflicts among dependencies and with other applications. A setup.py file is provided in the root of the project for an automatic installation, which requires executing:

```
python setup.py install
```

Also in the repository there are some tips for specific libraries.

The basic libraries to install are:

- requests==2.5.3
- tabulate==0.7.4
- pycurl==7.19.5.1
- httplib2==0.9
- simplejson==3.6.5
- goslate==1.4.0
- xmltodict==0.9.2
- beautifulsoup4==4.3.2
- futures==2.2.0
- six==1.3.0
- dill==0.2.2
- reportlab==3.1.44

The next libraries could have installation problems or special issues that need to be explained:

- beautifulsoup4

  This library is installed with dict.cc library, dict.cc is integrated in the project, that is, it doesn't require installation, but beautifulsoup4 does require.
- mstranslator

  Like dict.cc, this library is inside the project, the unique dependency is the library requests.
- pycurl and httplib2

  The first installation of these libraries were difficult for me because the pip installation can't decide a correct configuration for my computer, so I had to test different versions and finally these worked. Probably these libraries are already installed if you develop or use Python applications that need remote connections. If there are problems to install pycurl, try to install the next

libraries with:

```
sudo apt-get install libcurl4-gnutls-dev librtmp-dev python-dev
```

- Reportlab

  Could be problems with this PDF library if during the installation the arial.ttf font is not found. Sadly, there isn't a generic solution that solves this issue.

The installation steps are:

1. Clone the repository[1] or copy the provided repository in the pathos_installation directory.
2. Move to pathos_repository.
3. Execute python setup.py install
4. Depending on your configuration, you need to install the required libraries displayed by the error messages, in my case the pending libraries are pp and pyre/pythia (that are located in pathos_installation if you need them). Some compressed libraries can be found in pathos_repository/external. The installation process is the same, using the python setup.py install command. Additionally, pathos for Windows requires Visual C++ 9.0[2].

Finally, the installation process can be complex, specially pathos, so if the proposed version cannot be installed, please, consider the direct download using "pip" or "easy_install".

3.2.4 Documentation

Sphinx[3] must be installed to generate the documentation. Go to the docs directory and execute:

```
make html
```

After the generation, the mainpage is located at docs/build/html/index.html

3.2.5 Overview

The next sections explain all the necessary concepts to understand the architecture designed from the most basic concepts to the final implemented applications.
First of all they are explained the data structures used, after that, how the project can be configured,

[1] https://github.com/uqfoundation/pathos
[2] http://www.microsoft.com/en-us/download/details.aspx?id=44266
[3] http://sphinx-doc.org/

then how works the designed abstraction layer and which components are created to support the main module, and finally, how must be used the implemented scripts.

3.2.6 Translation utils

translation_utils.py contains the fundamental functionality of the whole application, which is used in all the processes in one way or another. It defines the fundamental entities TranslatorTask, TranslatorJob and Translation.
A TranslatorTask stores the text to be translated from a source language to a list of target languages. A TranslatorJob stores the name of the translator that made a translation, the list of translations as a dictionary and the time spent to generate them. The keys of the translation dictionary are language codes, and the values, lists with the translations. For example:

```
{
    "es": ["hola", "saludo"],
    "it": ["ciao"],
    "fr": ["bonjour", "salut"]
}
```

A Translation is the combination of a TranslatorTask and a list of TranslatorJob, that is, the modeling of a translation process, containing what is to be translated and the results for each involved translator. These three data structures have a __unicode__() function to show a representation of the data, it's important to use this function and not the default __str__() because the Unicode encoding is necessary in most of the cases. Python 3 solves this issue, but with Python 2 the solution is to use of the previous function or changing the default encoding configuration of Python, affecting all the Python libraries and applications.

3.2.7 Translator

The abstract class Translator defines the methods that a translator must implement, every implementation of a translator needs to extends this class for a correct behavior. The constructor only needs the optional parameter verbose to show information about the current state of the translations.

The abstract functions that each translator must implement are:

- get_translation: From a text, a source language and a target language, it returns a tuple where the first element is a list of translations (strings) and the second is the error code status. The error code can be an HTTP error returned by the translator service or an own code if a problem is detected, in general these are the error codes:
  - 200: Ok
  - 204: No content
  - 400: Bad request
  - 401: Unauthorized
  - 403: Forbidden
  - 404: Not found
  - 408: Timeout
  - -100: Not supported language
  - Plus other specific codes specific to each translation API

  In fact, the types of error codes are split in two: 200 (OK), where all gone correctly, and the rest, where the translation cannot be done.
- get_name: Returns the name of the translator.
- get_languages: Returns a set with the code languages (strings) that the translator supports.

The functions already implemented by Translator are:

- set_verbose: Changes the value of the verbose flag.
- translate: The most important function of this class. This function receives a TranslatorTask and obtains translations for each required target language using the get_translation function (implemented by translator subclasses). If the verbose flag is activated, it shows information using the standard error output about the current state of the translation and its result. After all the translations are done, the function returns a TranslatorJob instance.

3.2.8 Query wrapper

It facilitates calls to remote services, common data structures and functions that most of the translators use. However, some translators require specific calls which will be also presented.
The main library used is pycurl, a Python interface that use libcurl to make requests to Web services. Some translators work with the HTTP method GET and others with POST. So we have the methods

do_get and do_post, both methods need an endpoint, the headers, a callback to store the information and optional timeouts to abort the query if the server does not respond. Additionally do_get needs a dictionary with the parameters, and do_post needs a dictionary with the parameters if the encode flag is activated, or a string representation of the dictionary if encode is deactivated. Both functions return the response code of the server or a timeout if the server connection does not work properly.
Also there are a Buffer class that stores the response from the server, and JSONBuffer (subclass of Buffer) that transforms JSON content to a dictionary.
However, some translators that will be explained later use httplib2 and urllib for special reasons, so the implementation of each is directly done in the corresponding translator.

3.2.9 Transfusion

Currently, what has been introduced are just the data structures and the generic behavior of translators, so now we need an abstraction layer to work more easily and facilitate the integration with other programs or modules. Transfusion is a set of functions used to configure and launch translations, avoiding the headaches of the individual implementations of the translators and how the data is managed.
The constructor of Transfusion has some optional parameters: the list of translators, the verbose flag and which corrector will be used (as detailed in Section 3.2.10).
The functions provided by Transfusion are:

- set_translators: if we want to update a Transfusion instance with new translators.
- set_verbose: flag to enables debug information about the state and operation of the translations.
- get_translation: the main function, it initiates the remote connection with the translator service.
- get_correction: after receiving a Translation instance, this function applies the behaviors configured in the Corrector instance, returning a copy of the original Translation with the applied filters.

Also this class holds the list of supported languages, directly related with the languages required for NewdesksTranslations:

- English: en
- Spanish: es
- French: fr

- Portuguese: pt
- Chinese: zh
- Russian: ru
- Arabic: ar
- German: de
- Italian: it
- Dutch: nl

The default and expected source language would be English, with any other language unexpected results could be obtained.
As a curiosity, the name Transfusion is the combination of translation and fusion, defining perfectly what does this set of functions.

Concurrency

Notice the execution of the translators is sequential, but every translator behavior is independent from the rest, so an interesting feature is to benefit from Python management of pools of threads to parallelize their execution. The first implementation used the default multiprocessing library, but there was an issue related with the serialization of data from threads, which only works with function out of classes.

The current implementation uses pathos.multiprocessing, a library for concurrence with the capacity to serialize functions of classes using Dill. The function get_concurrent_translation does the same that the sequential function, replacing the iteration by a pool of threads (configurable by parameter) and doing a map, finally storing the job of each translator in an array.
Notice the execute function is a curious abuse of the Python language, the map function receives the execute function and the tasks, that are tuples where the first parameter is the translator instance and the second, the TranslatorTask. Therefore, the execute function for every thread receives one of these tuples, calling the first element with the function translate and sending the second as parameter, obtaining this code:

```
def execute(self, args):
    return args[0].translate(args[1])
```

Keys

The file keys.py stores the API keys, usernames and passwords for each translator, it is only used for a better organization of this information and as a configuration file that can be shared.

Settings

The file settings.py has a default configuration for the initialization of the translators using the provided keys and additional parameters if required. The instances of all the activated translators are stored in a dictionary, where the key is a simplification of the name of the translator and the value is the own instance.

Example of usage

At this point it is possible to call a translation using the previous Python components. This is a simple example of how it works.

First we need to create a TranslatorTask:

```
translator_task = TranslatorTask("hello", target_languages=["es", "it"])
```

Then, we create the Transfusion instance with the default translators, corrector and with the verbose flag activated:

```
transfusion = Transfusion(verbose=True)
```

We launch the translation process:

```
translation = tranfusion.get_translation(translator_task)
```

During this execution we can see the verbose for each translator:

```
Baidu working...
Baidu hello es 200 6.95918679237
```

```
Baidu hello it 200 1.03975510597
Google working...
Google hello es 200 0.194092035294
Google hello it 200 0.0981569290161
...
SysLang working...
SysLang hello es 200 3.33597993851
SysLang hello it 200 3.34506893158
```

Finally we obtain the translation and for example we can show the results for the first job:

```
print translation.jobs[0].__unicode__()
```

Obtaining:

```
Translator: Baidu Translations: Spanish: {Hola}; Italian: {Ciao}
Execution time: 7.99939799309
```

The Translation data structure becomes a massive structure fully of information and difficult to show. For a programmer it might be enough, but if a user wants to use it, probably a more user-friendly presentation would be useful, in the next sections we will see how this issue is solved.

3.2.10 Corrector

Corrector works as an extension for Transfusion, fixing and deleting incorrect translations, it is called with the get_correction function. The majority of the translators do not return exactly what is searched or expected. There is a list with the most common issues with their outputs:

- Same as input
- First letter capitalized
- All the letters capitalized
- Incorrect punctuation symbols, including unconnected pair symbols (only initial exclamation for example)

- Words inside brackets
- Absolutely incorrect text

Lots of these translation are really just variations of the correct answer, for example, the term hello in Spanish is "hola", but translators output also includes "Hola", "hola.", "HOLA" and "hola (saludation)". However, in order to process translators output in an aggregated way and decide the best translation, their output needs to be formalized to some extent and make all the previous outputs equivalent to just "hola".
Therefore, once these variations were collected, Corrector was implemented to try to minimize their impact. It was designed taking in mind the capacity to add filters as modules, including the possibility to execute these individually, so it is easily extensible to new identified variations.

3.2.11 Filters

All the filters derive from the abstract class AbstractFilter, whose unique method is apply, which receives a term as parameter. Alternatively, this function can be implemented using **kwargs (parameters as a dictionary) if more parameters are needed.

The provided filters are:

- LowercaseFilter: It has different behavior depending of the input:
  - First letter capitalized and the rest in lowercase → First letter in lowercase
  - All the letters capitalized → All the letters in lowercase
- SameFilter: Compares the input term with the translation, if they are the same it returns a void string. This filter does not follow the filter interface because its apply function needs two parameters, therefore this filter is directly integrated in the corrector.
- ParenthesisFilter: Removes the characters between brackets, and also the brackets. In the constructor it can be indicated the regular expression to use.
- PunctuationSymbolsFilter: Removes the defined characters. In the constructor they can be indicated using a string.
- StripFilter: Removes spaces, tabulation and other separators on the left and on the right of the input string. In the constructor it can be indicated which characters are used as string.

Unused and proposed filters

Other filters have been created but are currently not used because they do not fit the output of the current translators, though might be useful in the future:

- LengthFilter: If the term exceeds a specific length, the translation is discarded.
- ProportionFilter: If the translation length exceeds X times the original term length, it will be discarded.
- ArticlesFilter: Removes common words that can appear near nouns, for example articles, but can be used for any type of words like prepositions and conjunctions, every language needs its own set of words.
- StructureFilter: For particular patterns that invalidate translations, for example, (`<error code>`) text.
- AlphabetFilter: If the alphabet used by the translation differs from the correct alphabet, for example, an Arabic translation with Latin letters.
- GrammarFilter: Similar to StructureFilter, it identifies grammar rules that all the words of the same type must comply, if for example the source terms are gerunds and are known that the end of the translation must be a specific sequence of letters, these can be detected (for English is -ing, for Spanish is -ndo)

Filters execution

The file corrector_configuration.py contains the used filters and the available languages. The default_filters ordered dictionary is the structure sent to the corrector to apply the filters, the value for every string key must be a list of AbstractFilter instances. This special organization of the data has sense if the apply_filters and apply_all functions are checked in the Corrector class.
The OrderedDict must be provided in the constructor, if it is undefined, it will be used the default filters from corrector_configuration.py.
apply_filters iterates over the list of filters identified by the key filters_type from self.filters and returns the chained output after the filtering. This function can be executed individually if only a set of filters is required. apply_all sends the term to all the filters in self.filters in the specific provided order. Initially it is checked if the source term and the translation are the same (SameFilter behavior), after that are executed all the groups of filters and after each is checked again if the terms are equal, this

allows an early return response if this event occurs, for example, it could be a first group of fastest filters, another more slowly, and finally a very heavy filters that take a lot of time, if the equality of terms can be detected before the execution of the second and the third group, the performance will be better.

The expected workflow of the corrector is the creation of an instance of Corrector with the selected filters and languages and the execution of the apply_all function. All the filters should extend the AbstractFilter super class, because these will be easily integrated and executed in the same way. If some extra filters with other needs are required, these can be called individually by the programmer.

3.2.12 Writers

To provide more useful and usable translations output a set of writers have been developed. Some are intended for users and others to serialize the information in different formats.

There is not a common interface because every translator can use different data and optional flags, so every implementation is unique and has different parameters. If the number of writers was higher, probably a common interface with kwargs or the use of class hierarchy would be good options.

Write utils

The file write_utils.py contains a set of functions used by all the writers, and also specific functions for the writers that generate tables. These are the most important functions:

- get_write_file_descriptor: Every writer must call this function to obtain a file descriptor of a specific file to store the generated data, the available strings are:
  - "stdout": returns sys.stdout, the text will be displayed in the screen.
  - "default": returns a file descriptor at the root whose name is the current date, the term and the source and target languages.
  - A path: First it is checked if the path exists, if not, it is created. It returns a descriptor file whose file is inside this path and whose name is the previous default name explained.
  - A file: Returns the descriptor file of this file.
- write_utf8_encoded: Encodes the input text using UTF-8 and writes it in the file descriptor.
- close_file_descriptor: Closes the file descriptor unless it is the standard output.

JSON

The most important format to serialize the translations. It is used by other applications because is easy to dump it into a dictionary, for a normal user it is readable, but uncomfortable.

XML

Another useful and common format to serialize the data. Using the xmltodict library, these type of files can be loaded as a dictionary, like JSON.

PDF (experimental)

It can be used to store translations in a more usable format. The library used to generate the PDFs is reportlab. This writer is experimental, it is only created to demonstrate the possibility to create automatic reports with the translations.
The method used to write text uses the low-level functions of the library to work correctly with Russian and Arabic languages, that is, all the text positions and page breaks are managed by the custom implementation and not by the library. Finally, there is an issue with Chinese characters so it is not possible to generate PDF with terms in this language. They will be just showed correctly if the user has the Asian Language Pack or TrueType fonts with Asian characters installed and configured.

Standard output

This is the first implemented writer to test Transfuse. It has problems if the width of the terminal is not wide enough. Then, if the translation table is misplaced, the solution is to store the output in a file and view if with and editor featuring a disable word wrap flag. The library used to create the tables is tabulate[4], a Python module created by Sergey Astanin to display tables. It supports the following formats: fancy_grid, grid, html, latex, latex_booktabs, mediawiki, orgtbl, pipe, plain, psql, rst, simple and tsv.
This library has a reported issue[5] with the Chinese characters, so when the bug is fixed, tabulate will be updated to the last version.

[4] `https://bitbucket.org/astanin/python-tabulate`
[5] `https://bitbucket.org/astanin/python-tabulate/issue/51/chinese-characters-using-fancy-grid#comment-None`

Standard individual output

To solve the previous issue with the misplaced table, another display based on tables (also using tabulate) was implemented. This representation shows the same task table, but now every language has an individual table, so the output is narrower as opposed to the wide tables generated by the standard output formatting.

Raw output

The most basic writer, it simply shows the __unicode__() output. It could be useful to store a record of translations, but for translations with a lot of information it is difficult to read.

3.2.13 Transfuse

Transfuse is a command-line application that uses Transfusion and the writers to provide a more usable command-line user interface that allows users to request and store translations. All the parameters are optional and the following instructions are provided to guide users.

Arguments

- `-t | --term`: Text to be translated, it's recommended the use of quotes to avoid problems with spaces. By default is "hello", only used for testing.
- `-sl | --source-language`: Language code provided by Transfusion. By default it is English.
- `-tl | --target-languages`: List of language codes provided by Transfusion. If this argument is not present, it's assumed all the language will be used except the source language.
- `-tr | --translators`: List of names of the translators. The names are obtained using the default translators dictionary keys from settings.py. If this argument is not present, it's assumed all the default translators will be used.
- `-so | --standard`: Output the tasks in a table and the translators for each language in another.
- `-soi | --standard-individual`: Output the tasks in a table and each language has a table, it's more readable in some cases.
- `-j | --json`: Output the JSON representation of the translation.
- `-x | --xml`: Output the XML representation of the translation.

- `-p | --pdf`: Output the PDF representation of the translation, it must require a parameter, that is the filename where the data will be stored.
- `-ro | --raw-output`: Output the default Unicode representation of the translation.
- `-s | --show`: Offer specific help about a topic, the options are:
  - translators: Show the available translator names for -tr.
  - languages: Show the available language codes and its name for -sl and -tl.
  - table-formats: Show the available table formats provided by tabulate.
  - all: Shows all the previous information.
- `-v | --verbose`: Active the verbose flag for Transfusion.
- `-c | --correct`: Active the correction flag for Transfusion.
- `-h | --help`: Show the help provided by argparse.

Special arguments compatible with `-so | --standard`:

- `-ti | --time`: Add a new column with the total time execution and the time for each translator.
- `-td | --time-decimals`: Change the number of decimals for a best fit.

Special arguments compatible with `-so | --standard` and `-soi | --standard-individual`:

- `-tf | --table-format`: Change the table format used by tabulate. By default is "fancy_grid".

Arguments for concurrency:

- `-co | --concurrent`: Active the concurrent execution of the translators.
- `-nt | --num-threads`: Set the number of threads used for the concurrent execution.

Additional information

The writers can be combined in any desired way, all the options that involve outputs need zero or one parameters (except PDF that requires one). If no parameter is set, the data will be displayed in the screen. If a parameter is set, it must be a filename, a path (it must end with a slash '/'), "stdout" or "default", as explained in the write utils section. At least one output option must be provided, otherwise the data will be lost after the execution ends.

The verbose messages are only displayed on standard error, that is, these information does not interfere with the translation information stored in files. From Transfusion, the file name is automatically chosen.

Transfuse is the act of doing a transfusion, in this case, Transfuse canalizes a Transfusion from the memory of the computer into another entity, either the screen, a file, or the user who is seeing the

translations.

Examples of usage

There are some examples that can help to understand how Transfuse must be called. Show help:

```
transfuse --help
```

Show available options:

```
transfuse --show all
```

Show available translators:

```
transfuse --show translators
```

Translate "hello" from English to Spanish and Italian using all the translators, standard output and verbose:

```
transfuse --term "hello" -tl es it -so -v
```

Translate "dog" from English to Russian using Google, Yandex and Bing, standard output individual with tsv table format and with the corrector:

```
transfuse --term "dog" -tl ru -tr google yandex bing -soi -tf tsv -c
```

Translate "cat" from English (explicit) to German using Google, OneHourTranslation, SDL and Hablaa, standard output, JSON output in the file "json_test", XML output with default filename in the folder "new_directory", raw output in a default filename, with concurrency and 4 threads:

```
transfuse --term "cat" -sl en -tl de -tr google onehour sdl hablaa -so --json
json_test --xml new_directory/ -ro default -co -nt 4
```

3.2.14 Grouper

Previously we have seen that using transfuse the user can obtain the required translations in different formats. If this information has a low grade of relevance or only a few translations, it's better the use of the table outputs, but if a big set of information is obtained and this must be stored an checked later, the best option is the JSON format, with a medium grade of readability for humans, but perfectly understandable by machines.

The decision to implement this application comes after the generation of the translations of the pest words for the filters. The creation of this quantity of files can be tedious, but only takes time, transfuse (with the help of transfusion) does all the job, the great problem becomes at the time someone see the results, virtually intractable using the current table outputs, that are nice for a few words and languages in a particular moment.

The main task of grouper is to parse lot of information to display it based on the requirements and chosen options by the user, offering three types of output and two criteria.

Configuration file

First of all, the configuration file can be optional, but without sources or scores, the application loses all its sense, using all the time a default source with a score of 1, so it's extremely recommended the use of this. The first step to use grouper is the creation of a JSON file with the configuration, the structure is the next:

```
{
    "default": {
        "default": 50,
      "<language code 1>": 40,
      [...]
      "<language code n>": 30
    },
    "sources": {
        "<source 1>": {
            "default": 60,
            "<language code 1>": 70,
```

```
            [...]
            "<language code n>": 80
        },
          [...]
        "<source n>": {
            "default": 60,
            "<language code 1>": 65,
            [...]
            "<language code n>": 50
        }
}
```

The "default" key must be present with an inner "default" key, with more specific scores for the language codes. The "sources" key stores the scores for each source, all the sources used in the translation files must be present or they don't be counted. If a language is not found, it will be used the default value.

A little example of a configuration file is:

```
{
    "default": {
        "default": 50
    },
    "sources": {
        "Google": {
            "default": 60,
            "es": 70,
        "fr": 40
        }
```

Outputs

There are three different report types, each of these is specially created for a specific purpose discovered during the development. The three outputs are formatted using TSV, so they can be opened with OpenOffice Calc.

Individual term report

For each term and for each language is showed the score and the number of sources. This report is designed to verify the program is working correctly, that's for this the extension is .info and not .tsv, because it is only an informative report of the execution, providing unnecessary information for a real user.
Example:

insects			
	es		
		insectos	
			No. sources: 5 Score: 315
		los insectos	
			No. sources: 2 Score: 120

Table 3.1: Grouper - Individual term report example

Translations report

This report could be a classical file used by a professional human translator, but instead of containing only one translation, the user can decide how many wants. The languages are displayed horizontally and the terms vertically, so if for example, a user only wants a language, he can export this copying it in another spreadsheet. The displayed languages are the languages that at least one term has as a translation, that is, if for example the user only wants Spanish and Italian, the translation files only must contain these languages. The order of the translations is based in their score or number of sources, the user can chose which method prefer, in general, the two methods are correlated, but if the configuration file has big differences in the scores, the results will vary.
Example:

insects	it	es
	insetti	insectos
	Insetti	los insectos
	gli insetti	

Table 3.2: Grouper - Translations report example

Language reports

The translations report is very useful to see the translations in all the languages to use these, but what happens if the user wants to verify these translations? This set of reports solves the problem, for every language is created a file with all the terms and their translations, but the translations column is duplicated, the left column has the original translations and the right will be used to modify these and see the changes. When the files have been corrected, a parser can check the differences and inform of these. Example:

insects	es	es
	insectos	insectos
	los insectos	los insectos

Table 3.3: Grouper - Language reports example

Arguments

- `-s | --sort`: Criterion used to sort the translations. Options: source, score
- `-i | --input`: The JSON file or path (with JSON files) where are found the translations
- `-c | --conf-file`: The settings about the sources and their scores
- `-v | --verbose`: Show information about the current state of the execution using the error output
- `-nt | --num-translations`: Number of maximum translations displayed for output and output-languages
- `-o | --output`: Must be a file. Write all the translations for each term and language in the TSV file
- `-ol | --output-languages`: Must be a path. Write a set of files with the translations inside the path, each file contains one language
- `-r | --results`: Write a detailed report about the number of sources and the score of each term for every language

Like transfuse, the three outputs: ouput, output-languages and results can be combined in the desired way, but one of this must be present.

Examples of usage

Creates a information file called results using the translation files inside translations_directory with the configuration file grouper_configuration.json:

```
grouper -i translations_directory -c grouper_configuration.json -r results
```

Creates a translation report called translations using the translation files inside translations_directory with the configuration file grouper_configuration.json, with verbose and the translations are sorted by score:

```
grouper -i translations_directory -c grouper_configuration.json -o translations -v
-s score
```

Creates a folder called languages_folder with the double translation columns using the translation files inside translations_directory with the configuration file grouper_configuration.json, with verbose and the translations are sorted by source, with 2 as a maximum number of translations per term:

```
grouper -i translations_directory -c grouper_configuration.json -ol languages_folder
-s source -nt 2
```

Practical case and error detection

After the implementation of Transfuse, the corrector and an initial version of the grouper were done, Roberto sent me a list of words needed for a new filter to detect new pests. This were a good moment to test the applications in a real scenario.

First try

The file provided contains 277 terms and are required all the supported target languages and the highest number of translations, that is:

$$277\ terms * 9\ target\ languages * 12\ translators\ =\ 29916\ petitions$$

There is a lot of work, if every petition takes 1 second the translation of all the terms takes 8 hours and 18 minutes, and remember, it was the first attempt to run Transfuse with this amount of data, a person must be in front of it and check if all is correct, and 8 hours is a lot of time.
A high percent of the terms are plants (cabbage, peach, pepper) and diseases and causes (mortality, risk, injury, tsunami), but also there are a some from other contexts like commercial brands and products (Google, Nokia, Microsoft), geographical places and buildings (volcano, Fukushima, factory farm), animals (pork, dog, sheep) and human terms (wars, smartphone, nuclear industry). All the words can be found at TFG/data/pest_filter_data/pest_words.txt.
At the time this test was done, all the translators are enabled, except Syslang/Frengly and iTranslate, the first because it has the problem with the timing petitions and the second because the limit of translations is very low. These are: Google, Bing, MyMemory, SDL, WorldLingo, Yandex, Yandex Dictionary, OneHourTranslations, Hablaa, Glosbe, dict.cc and Baidu.

Script

To translate all the words the next script was launched:

```
#!/bin/bash
while read line
do
   term=$line
   transfuse --term "$term" -soi pest_soi/ --json pest_json/ -v -c -tf tsv
done < $1
```

The script reads all the lines of the file provided in the first argument ($1) and calls Transfuse for each line/term, notice the last line must be void.
Transfuse automatically stores the TSV and JSON files into the pest_soi and pest_tsv directories, that is 277 files in each folder following the name formatting: `<date>_<term>_<source language>_<target languages>.<format>`.

Issues

Initially the .tsv extension is not properly wrote, all the outputs from -soi (and -so) were .txt, it was changed to avoid to modify the extensions every time, now if the -tf argument is tsv, the file extension is .tsv, other formats also supports this feature.
In the Hablaa's ToS is explained that the service is free, but if the same IP sends lots of petitions the services will be denied, sadly this occurred, the concurrent execution of the script provokes this issue, the regulation is by IP, so multiple users can perform translations if these use different IP or if the workload is low.
OneHourTranslation had a bug in the response status, the code is correct (a 0), but the field msg contains "General error" in a extremely lower number of cases. Extending the condition with this new field to decide if a petition is done correctly solves the problem.

Results

After some hours of executions and tests the final results are obtained, the methodology to obtain these are the execution of the script in three different consoles to obtain a good performance/time ratio and to avoid the 8 calculated hours, becoming in around 3, notice someone must be present during the executions to detect problems, almost this first time, every execution only had 10-20 terms, and when one ends, a new set of words was launched, this behavior also was done to detect unhandled exceptions and to have more control of what was doing.
Some files are checked and at first appearance (and as expected) there are good and bad translations, but more or less, all the translators tries to do a good job, Arabic probably was the lowest number of translations, were other languages have translations by all the translator, Arabic had absolutely nothing. Finally, the corrector does a good job, there were some fears about Russian, Chinese and Arabic but any issue was detected.

3.2.15 Validator

Validator was created to generate automatic reports about the accuracy of the translators and the success rate of the terms. Additionally, with these information can be created more realistic scores for grouper.

Configuration file

All the terms that will be validated must be present in this file, including the source and the target languages following the next structure:

```
{
    "languages": {
        "source_language": "<language code>",
        "target_languages": "[list of language codes]"
    },
    "valid_translations": {
        "<term 1>": {
            "<language code 1>": "[list of valid translations]",
            [...]
            "<language code n>": "[list of valid translations]"
        },
      [...]
        "<term n>": {
            "<language code 1>": "[list of valid translations]",
            [...]
            "<language code n>": "[list of valid translations]"
        }
}
```

A configuration file example can be:

```
{
    "languages": {
        "source_language": "en",
        "target_languages": ["es"]
    },
    "valid_translations": {
        "dog": {
```

```
            "es": ["perro", "can"]
        }
}
```

Outputs

There are a lot of configurations to classify the validations using the parameters translators, languages and terms. After some tests about which are better, two implementations were elected, one focused on the translators and another on the terms.

Report by translator

For every translator is showed the percentage of accuracy for each language, additionally with the detailed flag, they are displayed all the terms and their validation status.
Example without details:

Google		
	es	75% (75 of 100)
WorldLingo		
	es	55% (55 of 100)

Table 3.4: Validator - Report by translator example without details

Example with details:

Google			
	es	75% (75 of 100)	
		insects	True
		scratch	True
		kill	True
		lack juice	False
		[...]	
WorldLingo			
	es	55% (55 of 100)	
		insects	True
		scratch	True
		kill	False
		lack juice	False
		[...]	

Table 3.5: Validator Report by translator with details

Report by term

For every term is showed the percentage of accuracy for each language, additionally with the detailed flag, they are displayed all the terms and their validation status. Example without details:

sunflower		
	es	92% (12 of 13)
elm		
	es	61% (8 of 13)

Table 3.6: Validator - Report by term without details

Example with details:

sunflower			
	es	92% (12 of 13)	
		Baidu	True
		Google	True
		WorldLingo	True
		[...]	
elm			
	es	61% (8 of 13)	
		Baidu	False
		Google	True
		WorldLingo	True
		[...]	

Table 3.7: Validator - Report by term with details

Arguments

- `-i | --input`: Directory with the JSON files that contain the translations
- `-tr | --report-by-translator`: Do an analysis focused on the accuracy of the translators
- `-te | --report-by-term`: Do an analysis focused on the success rate of the terms
- `-vf | --validation-file`: The JSON file with the required information for the analysis
- `-v | --verbose`: Show information about the current state of the execution
- `-d | --detailed`: Show more information about the statistics
- `-s | --separator`: Add extra lines for a better readability
- `-p | --percentage`: The percentages are showed in a most readable style

Additional information

The difference between the use of percentage or not is the next:

With percentage:

61% (8 of 13)

Without percentage:

61 8 13

As it can be seen, the percentage option is more readable for humans, and without it, a parser can read every individual value because they are separated by tabulations.

Examples of usage

Creates a report about the accuracy of the translators, with detailed information about which terms are accepted, with readable percentages:

```
validator -i translations_directory -tr translators_validation -d -p
```

Creates a report about the success rate of the terms, with verbose, with additional spaces:

```
validator -i translations_directory -te terms_validation -v -s
```

3.2.16 Practical case

A list of words was created to verify and try to obtain good parameterizations of the correctness of every translator. The domain of the list is focused on plants and animals glossary, also including verbs, symptoms and plant parts. All the terms are enough known by everyone, it's true that some are strange, but because are related to the world of the plant diseases, but experts on this probably know these. These terms are translated by me to Spanish, the most of the terms are known by me, but others need additional investigation to know exactly what is the translation, obviously the use of machine translators is not allowed here, the translations were found using English-Spanish and Spanish-English dictionaries and specialized websites with pair translations done by humans, like Linguee[6].

[6] http://www.linguee.com/

Transfuse

The translations were obtained using the bash script presented in 3.2.14 that reads a file with the words and executes:

```
transfuse --term "$term" -soi translations/ --json translations_json/ --pdf
translations_pdf/ -v -c -tf tsv -co -tl es
```

The outputs are individual language (but only Spanish is used) with TSV formatting, JSON and PDF. The verbose flag is activated to check the execution, any issue was found. The corrector is activated. And finally, the concurrent execution of the translators is activated with the default number of threads. The translators used are: Google, Bing, MyMemory, SDL, WorldLingo, Yandex, Yandex dictionary, Baidu, Hablaa, Glosbe, Dict.cc, OneHourTranslation, SysLang. All except iTranslate.

Grouper

When all the translations were done, a configuration file was created containing the used translators, it can be found in TFG/data/spanish_data/grouper_configuration.json.
After that, grouper is called with the next command:

```
grouper -i translations_json/ -c grouper_configuration.json -o grouper_translations
-s source
```

Finally, the file grouper_translations.tsv contains all the translations ordered by source.

Validator

The validation configuration file contains all the terms in the wordlist, with the Spanish translations done by me. It can be found in TFG/data/spanish_data/validation_data.json.

The commands executed to obtain the reports are:

```
validator -i translations_json/ -vf validation_data -tr translators_report
```

```
validator -i translations_json/ -vf validation_data -te terms_report
```

In the same folder where is placed validation_data.json can be found the generated reports called translators_report.tsv and terms_report.tsv, among other additional reports.

Statistics

First of all, notice that the valid translations in the validator configuration file don't reach all the domains of the terms, for example, the term wet has some translations, the provided translations are: "humedecer", "húmedo", "mojar" and "mojado", but Bing translator returns "húmeda", a valid translation, so all the percentages really are higher, the more valid translations there are, the more percentages are obtained if the translators can really translate the terms. Also, the word cactus is the same in Spanish, so all the translators will fail because the corrector detect the same words.

The next graphic shows the percentage of success for each translator, it is created using the translators report:

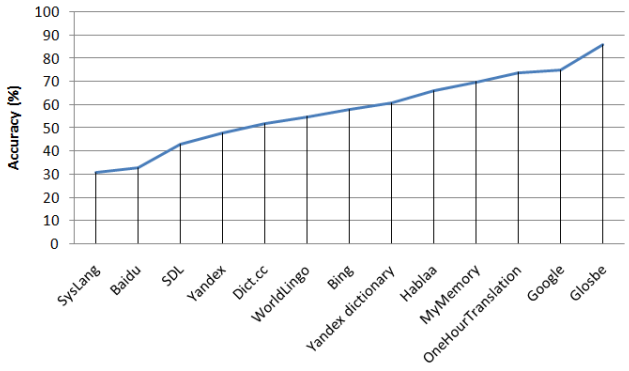


Figure 3.2: Translators accuracy

According to statistics, Glosbe is the best with a 86%, that is because is the translator that returns more translations, so, it works with brute force, using only Glosbe is not a good idea because a lot of concepts will be obtained. The intermediate translators go from Yandex Dictionary to Google, with a 60-75% success rate, these translators do a correct job with few results. From SDL to Bing, the success rate vary from 48% to 58%, approximately they find the valid translation half of the time, is not a bad result, but others do it better. Finally, Syslang and Baidu have the worst punctuation, 31% and 33%, respectively, probably users don't want to use these.

This graphic is created using the terms report, showing the number of parameters with the same percentage of success:

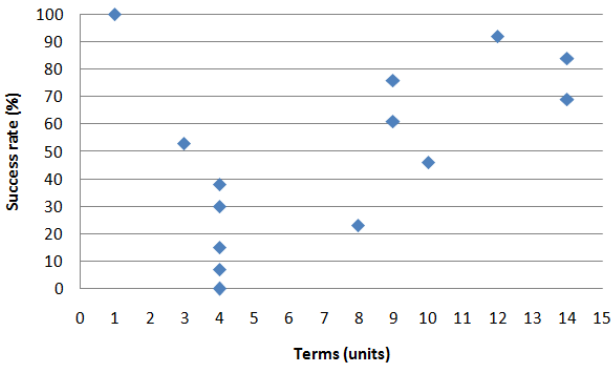


Figure 3.3: Translators accuracy

The tendency of the graphic is that the more terms are, the more accuracy have, so the most of the terms are grouped over the 50% of success rate. The terms with difficulties to be translated are composed words and verbs, so probably these words have correct translations, but the range of words is higher than the others, and not all the possibilities are configured as explained at the start of the analysis, so if a more exhaustive set of words the results would be better.

Conclusions

After these two analysis now we have a better knowledge about the global possibilities of Transfusion and which are the best translators to use on it. If we take the average of all the translators it is 57% of accuracy, but if we only use the best the average grows up to 72%, an taking in the account that valid translations aren't contemplated in the configuration file, this number might grows a little bit more.

3.2.17 Translators

During all the development were found translator services compatible with Transfusion, some of them are adapted projects created by the community, but most of them are directly implemented specially for Transfusion. In this section it is explained which languages are compatible, the characteristics of every implemented translator, including how are the inputs and the output, their limits and issues, and in some cases, the curl queries used for testing.

Available languages

The next table shows which languages supports every translator, some of them also has additional information about the number of translations available.

	English (en)	Spanish (es)	French (fr)	Italian (it)	Russian (ru)	German (de)	Portuguese (pt)	Dutch (nl)	Arabic (ar)	Chinese (zh)
Google										
Bing										
SDL										
MyMemory	749.550	177.971	249.022	106.489	73.454	335.157	103.969	198.688	31.823	45.058
WorldLingo										
Yandex										
Yandex dictionary										
Frengly (SysLang)										
OneHourTranslation										
Hablaa		227.016	290.442	171.390	35.328	1.434.103	68.750	75.131	91.877	190.188
iTranslate4										
Glosbe										
Dict.cc										
Baidu										

Figure 3.4: Available languages for each translator

The data for MyMemory were obtained on April 21 and for Hablaa on April 13.

Google Translate

Overview

Probably the most famous translator, its website offers pair translations near 100 languages, automatic source language detection, voice recording, phonetic pronunciation (text and audio), keyboards for all the languages, specially for non-ascii languages, like Chinese, entire website translation and finally a toolkit to upload our documents .

The machine learning is based in text comparison of the same texts in different languages, Google has its own book service[7], so one of the sources is the book comparison, and the translations of the books are done by humans, offering a massive number of quality translations.

[7] `https://books.google.es/`

API

That was the first translator used in the Django project, it was added by Josep Maria as a test for the translation process. This translator is wrapped in a Python library called goslate, there are multiple implementations, but the original comes from the user zhuoqiang[8] and his repository[9]. The library can be used as a command line application, but for our project it is directly imported.

Input

The most important method is translate, with the classical triplet of parameters term-target-source, also supports automatic detection for the source language provided by Google. The queries have the encoding parameter to UTF-8.

Endpoint	http://translate.google.com
HTTP method	GET
Content type	application/json
	client: Simulates another application that uses this service publically (set a)
	sl: Source language code
	tl: Target language code
	ie: Input encoding (set UTF-8)
	oe: Output encoding (set UTF-8)
	dt: Indicates if the query is a translation (t) or a romanization (rm), like Chinese character to latin alphabet (set t)
	q: Text to be translated

Table 3.8: Google input

Output

The library cleans the raw obtained text to fix issues with the encoding, providing an unicode string. This is the unique response, an unicode string if all goes well:

u'hola'

Otherwise the string is empty.

[8] `https://bitbucket.org/zhuoqiang`
[9] `https://bitbucket.org/zhuoqiang/goslate/`

Limit of use

There isn't a specific limit, but probably if a massive number of requests are done, Google can ban the IP address.

Issues

Checking the code, the original author has some issues about the user agent, he needed to use Mozilla 4.0 because Google forbids the default urllib2 agent.
Google translate provides good translations and also it's fast and free. But the fact Google has a monetization way for its translate service, really it's not a free service, if Google wants, it can close the current endpoint and only the users with API key will be able to access it.
The 8th April 2015 the service stops abruptly, nobody can do queries, the endpoint used by goslate (and the rest of libraries) was closed or changed, because truly the API offered by Google is a payment service, but these use "alegal" ways to call public endpoints that Google products use, like the automatic translator of Google Chrome, another method is the use of Google Sheets an do indirect calls, but it is extremely slow. The issue was solved by goslate's author, he noticed that a new parameter (dt) is added, he simply updated the query with this change, other users migrate to another translators, like MyMemory. In 2011 and 2012 an issue about the closing of the free services happened, so probably in the next months or years will occur again if the ToS of Google change.

Bing

Overview

Originally it was Altavista Translator, created by Digital Equipment and Systran (that has its own translator[10]) in 1997. Some years later, in 2003, Yahoo adquired the current owner of Altavista (Overture Services), and then, in 2008, it changed to Yahoo BabelFish (not be confused with BabelFish[11] from The BabelFish Corporation). Finally, in 2012, Yahoo BabelFish becomes the current Bing Translator managed by Microsoft.
In brief, currently there are Bing Translator from Microsoft, Systran (requires fee) and BabelFish

[10] `http://www.systransoft.com/`
[11] `http://www.babelfish.com/`

(without API access).

Bing Translator offers text translations, automatic source detection, website translation and translations from files. Also there are some Microsoft products like Skype and Office that integrate it. Finally, the API provides some methods for the developers to integrate it as you can see in the next section, the available interfaces are HTTP, SOAP, AJAX and REST.

The source of the translations comes from the same texts in different languages and statistical techniques, obviously, the base of all is the original Altavista Translator, so Microsoft integrates it in with its applications and adds more equivalent texts.

API

The user wronglink[12] created a wrapper[13] with all the options that the official AJAX API supports, that is, translations from a language to another, automatic detection of the source language, translations from files, text to voice and suggestion of better translations.

The wrapper needs credentials to access the service, that are the client ID and the client secret, that can be obtained in the Azure marketplace creating a new web application[14].

Input

Endpoint	http://api.microsofttranslator.com/V2/Ajax.svc/
HTTP method	GET
Content type	text/plain
Parameters	text: Text to be translated to: Source language code from: Target language code options: The content type (plain/text) and the domain of the text (currently it seems to be only "general")

Table 3.9: Bing input

The query type used only returns one translation, also exists another call[15] to obtain more translations based on a classification from 0 to 100 about the correctness of the translation. After some tests, it is easy to see that some translations near 100 are exactly the same translation of the individual call, but others are incorrect, and the translations near 0 don't have sense.

[12] `https://github.com/wronglink/`
[13] `https://github.com/wronglink/mstranslator`
[14] `https://msdn.microsoft.com/en-us/library/hh454950.aspx`
[15] `https://msdn.microsoft.com/en-us/library/ff512402.aspx`

The implementation has the two possibilities using the more_translations and max_translations parameters in the constructor, the default function is request only a translation and maximum 10 translations if more_translations is enabled.

Output

The wrapper only returns an unicode string if only a translation is requested:
u'Hola'
If we want more translations the output is a JSON object:


```
{
   "Translations":[
      {
         "Count":0,
         "Rating":5,
         "MatchedOriginalText":"",
         "MatchDegree":100,
         "TranslatedText":"computadora"
      },
      {
         "Count":2,
         "Rating":1,
         "MatchedOriginalText":"computer",
         "MatchDegree":100,
         "TranslatedText":"computadora"
      },
      {
         "Count":0,
         "Rating":0,
         "MatchedOriginalText":"computer",
         "MatchDegree":100,
         "TranslatedText":"ordenador"
```

```
        },
        {
            "Count":0,
            "Rating":0,
            "MatchedOriginalText":"computer",
            "MatchDegree":100,
            "TranslatedText":"equipo?"
        },
        {
            "Count":0,
            "Rating":0,
            "MatchedOriginalText":"computer",
            "MatchDegree":100,
            "TranslatedText":"computadora"
        },
        {
            "Count":0,
            "Rating":0,
            "MatchedOriginalText":"computer",
            "MatchDegree":100,
            "TranslatedText":"Computadora;"
        },
        {
            "Count":0,
            "Rating":0,
            "MatchedOriginalText":"computer",
            "MatchDegree":100,
            "TranslatedText":"Computadora"
        }
    ],
    "From":"en"
}
```

This structure is modified for a better comprehension, replacing the unicode strings to readable text. The unique valid information is every value in TranslatedText of the Translations array. The Count, Rating and MatchDegree keys don't have the enough quality to be useful.

Limit of use

With the free subscription[16] there are 2.000.000 input characters per month.

Issues

An initial connection with the Microsoft's servers is done to obtain an access token, this token has an expiration time of 10 minutes, the used library manages this issue, but it is an increment of complexity and possible problems.
Currently the unique context is "general", probably in the next months or years new contexts will be added.
The translations from the query that returns more than one result has extremely low quality, these aren't validated by nobody.

SDL

Overview

SDL provides machine and human translations. The online translation site managed by SDL is FreeTranslation[17], where the users can translate texts and documents, also rate and hear these. Also there is an application for Android[18] and iOS[19]. Additional solutions[20] related to translations are offered to speed up the development of webpages and programs in multiple languages.

[16] `https://datamarket.azure.com/dataset/bing/microsofttranslator`
[17] `http://www.freetranslation.com/`
[18] `https://play.google.com/store/apps/details?id=com.sdl.translate`
[19] `https://itunes.apple.com/us/app/sdl-translate/id720669507`
[20] `http://www.sdl.com/cxc/all-products.html`

API

The API key can be obtained after the registration process[21]. This key must be set in the authorization header. The documentation[22] explains how to perform a curl query and how is the output.

Input

Endpoint	https://lc-api.sdl.com/translate
HTTP method	POST
Content type	application/json
Parameters	text: Text to be translated from: Source language code to: Target language code

Table 3.10: SDL input

Curl

The curl call has the next appearance:

```
curl -X POST -H "Content-type:  application/json" -H "Authorization:
LC apiKey=<PRIVATE_KEY>" -d '{"text":"hello", "from":"eng", "to":"spa"}'
'https://lc-api.sdl.com/translate'
```

Notice the post data is a string, not a key-value resource.

Output

The response is a JSON object with the original provided data and information of the translation. The useful field is translation.

```
{"from": "eng",
"charCount": 5,
"wordCount": 1,
```

[21] https://languagecloud.sdl.com/translation-toolkit/sign-up
[22] https://languagecloud.sdl.com/es/translation-toolkit/api-documentation

"to": "spa",
"partialTranslation": false,
"translation": "Hola"}

Limit of use

Taken directly from the documentation[23]: "For Machine Translation you can use up to 500,000 characters per month in Sandbox for free."

Issues

The post fields are really a string that's looks like a JSON object, and this string mustn't be encoded because it will be managed by the server, SDL works with the language code ISO 639-2/B[24], where the codes have three letters, using a dictionary with the equivalent codes solves the problem.

MyMemory

Overview

The site[25] has a little demonstration to test the capacities of MyMemory, the users can translate texts from a language to another, obtaining a machine translation if the text is a phrase or a specific term, and a list of translations with the most matched source text and its equivalent, these translations come from another webpages like Wikipedia.
Also the users can add translations, and vote, suggest and correct the existing.

The source of the translations can be obtained at the end of the translation page: "The Credits - Computer translations are provided by a combination of our statistical machine translator, Google[26], Microsoft[27], Systran[28] and Worldlingo[29]."

[23] https://languagecloud.sdl.com/translation-toolkit/api-documentation
[24] http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
[25] http://mymemory.translated.net/
[26] http://translate.google.com/
[27] http://www.microsofttranslator.com/
[28] http://www.systransoft.com/
[29] http://www.worldlingo.com/

API

Aside from translations, the API supports uploads with new translations, these can be directly sent directly with GET or uploading a TMX[30] file with a set of translations. When a translation is sent, also can be specified a subject[31].
The API key is not required, but if we are interested, we must register[32] and do a query to
`http://api.mymemory.translated.net/keygen?user=username&pass=password`
with the corresponding username and password. The output is a JSON object with the key.

Input

The documentation[33] is enough explicit and detailed to understand how works this translator.

Endpoint	http://api.mymemory.translated.net/get?
HTTP method	GET
Content type	application/json
Parameters	q: Text to be translated langpair: String that encodes the target and the source language as `<source_lang>-<target_lang>` key: Personal API key de: Email

Table 3.11: MyMemory input

Output

The response is a JSON object split in two parts, translations by users, and machine translation.

```
{
   "matches":[
      {
         "created-by":"Matecat",
         "usage-count":42,
         "reference":"",
```

[30] `http://xml.coverpages.org/tmxSpec971212.html`
[31] `http://api.mymemory.translated.net/subjects`
[32] `https://www.translated.net/top/?ref=mm`
[33] `http://mymemory.translated.net/doc/spec.php`

```
        "segment":"hello",
        "create-date":"2015-02-10 00:18:22",
        "last-updated-by":"Matecat",
        "tm_properties":"",
        "translation":"Hola",
        "last-update-date":"2015-02-10 00:18:22",
        "quality":"80",
        "id":"465122672",
        "match":1,
        "subject":"All"
    },
    {
        "created-by":"Matecat",
        "usage-count":1,
        "reference":"",
        "segment":"Hello",
        "create-date":"2015-04-10 18:34:41",
        "last-updated-by":"Matecat",
        "tm_properties":"",
        "translation":"Hola sdasda",
        "last-update-date":"2015-04-10 18:34:41",
        "quality":"74",
        "id":"465921498",
        "match":0.97,
        "subject":"All"
    },
    {
        "created-by":"Matecat",
        "usage-count":1,
        "reference":"",
        "segment":"Hello?",
        "create-date":"2015-04-14 01:14:57",
```

```
            "last-updated-by":"Matecat",
            "tm_properties":[
                {
                   "type":"x-project_id",
                   "value":"86047"
                },
                 {
                    "type":"x-project_name",
                    "value":"series_2"
                 },
                 {
                    "type":"x-job_id",
                    "value":"101671"
                 }
            ],
            "translation":"?‘Bueno?",
            "last-update-date":"2015-04-14 01:14:57",
            "quality":"74",
            "id":"465956537",
            "match":0.96,
            "subject":"All"
}
],
"responseData":{
"translatedText":"Hola",
"match":1
},
"responseDetails":"",
"responseStatus":200,
"responderId":"236"
}
```

As you can see, with a simple text like hello, some translations are strange and buggy, these kind of translation are direct uploads of the users, using the fields created-by and quality can be decided which grade of correctness is required.

The Python implementation can be configured in the constructor, deciding the minimum required quality (80 by default), the allowed authors (Wikipedia and Matecat[34] by default) or if all the authors are allowed (False by default). Also there is the field segment, that shows the source term which is obtained the translation.

If a translation is not available, the term will be translated using the machine translators named before, the created-by field is MT!.

The translatedText field is the first translation of the array of matches, independently this is a good or bad translation, that is, the responseText field is not useful, if there was any criteria that could decide which is the best translation taking care the quality parameter for example, it could be useful, but this is not the case.

Limit of use

Without providing a valid email in the field "de" it can be made 1.000 requests per day. Providing an email the requests per day increase up 10.000. More information can be found in the API limits page[35].

Issues

The translation of hotel from English to Spanish is "Por favor, especifique dos idiomas diferentes" (Please, specify two different languages), with the rest of language this issue doesn't happen. This translation is a user translation with a quality of 74, the correct translation hotel has 80, but the translatedText field has the incorrect, so we cannot trust that the best translation is placed in this place.

The JSONBuffer callback fails, the unique solution found is using a stream from the standard library io, replacing the buffer with a BytesIO buffer.

The use of the API key has a worse performance than without such. A simple example with the term

[34] `http://www.matecat.com/about/`

[35] `https://mymemory.translated.net/doc/usagelimits.php`

hello and all the supported target languages shows the issue:

Without API key (s)	With API key (s)
2.29422187805	11.8181540966

Table 3.12: MyMemory key comparison

The API calls are up to 415% slower than the same queries without API.

WorldLingo

Overview

Worldlingo is one of the oldest translators (created in 1998) on the Internet, it provides machine and professional translations for over 141 languages. The free machine translation services are text translation, document translation, web translation, email translation, all found in the main page[36].
Also there are some external services as bilingual dictionaries, translations about a specific vocabulary's job, localization of software and multimedia and mobile solutions.

API

There is a simple page[37] to test the possibilities of WorldLingo API, including encoding, output format, custom dictionary and error output location.
All the information about the API can be found in this document[38].
It includes a set of glossaries can be used to obtain better translations depending of the translation context.
The implementation is using the default API key.

Input

There is the decided configuration for the queries:

[36] `http://www.worldlingo.com/es/`
[37] `http://www.worldlingo.com/demo/api-html/WorldLingoAPI-HTMLDemo.html`
[38] `https://www.worldlingo.com/en/downloads/ServiceAPI.pdf`

Endpoint	http://www.worldlingo.com/S000.1/api?
HTTP method	GET
Content type	text/plain
Parameters	wl_data: Text to be translated wl_srclang: Source language code wl_trglang: Target language code wl_password: API key (secret by default) wl_gloss: Glossary or context of the translation (General by default)

Table 3.13: WorldLingo input

Output

The default and used configuration is that the data is directly displayed in the body of the returned HTML page following the next structure:

error code

translation

If all goes right the error code must be 0, otherwise the translation can't be done through a mandatory parameter is incorrect through the server side has an internal problem.

There is an example:

0

hola

The entire response must be decoded to UTF-8 because it is directly the body of the HTML document.

Limit of use

According with the documentation 1000 requests per month are permitted using the free service.

Issues

The output format could be better.

OneHourTranslation

Overview

OneHourTranslation provides human translations as first service, including translation of documents and multimedia, proofreading, transcriptions, translation of emails, webpages and applications. The free machine translator for users currently is not working, so for now the unique access to it is the API.

API

It has one of the best documentations, there are the web documentation[39] and a downloadable file[40]. The queries need the user account and the public and private keys, this information can be found after the registration in this link[41]. There is also a sandbox scenario with its specific keys, but it is designed only for testing.

Input

Endpoint	http://www.onehourtranslation.com/api/2/mt/translate/text?
HTTP method	GET
Content type	application/json
Parameters	source_content: Text to be translated source_language: Source language code target_language: Target language code account: User account secret_key: Provided secret key public_key: Provided public key

Table 3.14: OneHourTranslation input

Curl

This is the provided curl example in the documentation:

[39] http://www.onehourtranslation.com/translation/api-documentation-v2

[40] http://oht-webcontent.s3.amazonaws.com/resources/One_Hour_Translation-API-V2-implementation-guide.pdf

[41] https://www.onehourtranslation.com/profile/apiKeys

```
curl --request GET "http://www.onehourtranslation.com/api/2/mt/translate/text?
secret_key=<secret_key>&public_key=<public_key>&source_language=en-us
&target_language=fr-fr&source_content=text for translation"
```

Output

The service returns a JSON object split in three parts: status, errors and results.

```
{"status": {"msg": "ok", "code": 0},
"errors": [],
"results": {"TranslatedText": "hola"}}
```

The translation is placed in the TranslatedText field. A translation is correct if the field code is 0. If the translation can't be done the code remains being 0. There are a large number of error codes, most of them related to malformed queries or other actions which aren't direct translations.

Limit of use

Apparently there isn't a defined limit, but probably if a massive number of queries are done, the API key will be banned.

Issues

The character ´ (simple quote) is not decoded properly, it is displayed as the HTML character &#39, the HTMLParser library solves the problem.

Yandex

Overview

Yandex is the Russian Google, it is a set of search services (images, videos, maps) and it also has a translation tool. The translator page offers translations between languages, automatic source language

detection, textual and audio pronunciation and webpage translations. Also there is an application for Android, iOS and Windows Phone. The mechanisms used by this service for the translations are comparisons between texts and statistics using BLEU[42] metrics.

API

The documentation[43] is simple but well explained, it can be found which languages and parameters are available, and the different outputs and error codes depending the format. The API key can be obtained after the registration, requesting it and found it here[44].

Input

There are XML and JSON (and JSONP) formatting, the used is the JSON because most of the translators use this.

Endpoint	https://translate.yandex.net/api/v1.5/tr.json/translate
HTTP method	POST
Content type	application/x-www-form-urlencoded
Parameters	text: Text to be translated lang: String that encodes the target and the source language as `<source_lang>-<target_lang>` key: Private key

Table 3.15: Yandex input

Curl

The curl call has the next appearance:

```
curl -X POST 'https://translate.yandex.net/api/v1.5/tr.json/translate?
key=<PRIVATE_KEY>&lang=en-es&text=hello'
```

[42] http://en.wikipedia.org/wiki/BLEU
[43] https://tech.yandex.com/translate/doc/dg/concepts/About-docpage/
[44] https://tech.yandex.com/keys/

Output

One of the shorten and simplest JSON objects. The translation can be found in the text field.

```
{"lang": "en-es", "text": ["hola"], "code": 200}
```

The array of translation must contains one value because only one text is requested.

Limit of use

From the Terms of Use[45]: "Small translation projects, which do not exceed 10,000,000 characters a month (and no more than 1,000,000 in 24 hours), can use this service for free."
Additionally from the general Yandex ToS:
"2.3. Yandex reserves the right to set any limits and restrictions, including but not limited to those stated below: the volume of the text translated: 1,000,000 characters per day but not more than 10,000,000 per month."

Issues

Difficulties to find the content type for the pyCurl queries. Notice the curl call doesn't need this header. Erroneously, if it is incorrect the service returns the error 'API key is incorrect' instead of a better error message about the content type.

Yandex Dictionary

Overview

This is another service from Yandex, in this case Yandex Dictionary is a set of bilingual dictionaries between two languages created and extended from the translations of the Yandex translator. This service isn't available for normal users, it only be callable from the API.

[45] `https://translate.yandex.com/developers`

API

The documentation[46] explains the accepted language pairs and how are done the queries and which outputs are returned.

The process[47] to obtain the API key is the same as Yandex translator.

Input

The format types can be JSON and XML, the used is JSON.

Endpoint	https://dictionary.yandex.net/api/v1/dicservice.json/lookup?
HTTP method	GET
Content type	text/plain
Parameters	text: Text to be translated lang: String that encodes the target and the source language as `<source_lang>-<target_lang>` key: Private key

Table 3.16: Yandex Dictionary input

Output

For this translator the term to test is "computer" because it has synonyms.


```
{
   "head":{

   },
   "def":[
      {
         "text":"computer",
         "tr":[
            {
               "text":"ordenador",
```


[46] https://tech.yandex.com/dictionary/doc/dg/concepts/About-docpage/
[47] https://tech.yandex.com/keys/get/?service=dict

```
"syn":[
   {
      "text":"computadora",
      "pos":"noun",
      "gen":"f"
   },
   {
      "text":"Computer",
      "pos":"noun",
      "gen":"m"
   },
   {
      "text":"computador",
      "pos":"noun",
      "gen":"m"
   },
   {
      "text":"informatica",
      "pos":"noun",
      "gen":"f"
   }
],
"mean":[
   {
      "text":"laptop"
   },
   {
      "text":"desktop"
   },
   {
      "text":"computing"
   }
```

```
            ],
            "pos":"noun",
            "gen":"m"
         },
         {
            "text":"informatico",
            "pos":"adjective",
            "mean":[
               {
                  "text":"computing"
               }
            ]
         }
      ],
      "pos":"noun",
      "ts":"[pronunciation]"
   }
]
}
```


This JSON structure is modified in order to being more readable. Not just a set of translations are obtained, also the type of word, genre, pronunciation and other synonyms are retrieved.
The translations can be found in the text fields of def, tr and syn, except those within mean field.

Limit of use

From the general ToS: ”2.4. Yandex reserves the right to set any limits and restrictions, including but not limited to those stated below: the number of references to the Service: 10,000 references per day.”

Issues

The JSON structure changes depending of the term, this provoke that some test needed to be performed to avoid problems accessing the parsed dictionary from the obtained JSON.

Frengly (Syslang)

Overview

When this translator was discovered it's name was Syslang, but currently it's Frengly.
Frengly collects translations from users using a friendly interface, also can be added translations from source provided texts by the site. The translation page[48] supports translations between languages and autodetection of the source language.
Frengly's engine has a particular system[49] to rate the quality of the translations following different criteria about if a word is translated by a human or a machine, if there isn't a translation and depending the length of the translated segments.

API

The documentation[50] explains how do the queries, also it is supported a JQuery function to translate from another webpage.
There isn't API key for this service.

Input

The type format can be XML or JSON, but if an error occurs, regardless the chosen format, it is XML. Initially it was implemented with JSON, but this issue provoke the change to XML.

[48] `http://www.frengly.com/#!/translate`
[49] `http://www.frengly.com/#!/quality`
[50] `http://www.frengly.com/#!/api`

Endpoint	http://frengly.com?
HTTP method	GET
Content type	text/xml
Parameters	text - Text to be translated src: Source language code dest:Target language code email - Frengly account email password - Frengly account password outformat - Format of the response [xml/json] (set xml)

Table 3.17: Syslang input

Output

```
<?xml version="1.0" encoding="UTF-8"?><root>
  <text>hello</text>
  <translation>hola</translation>
  <translationFramed>hola|</translationFramed>
  <missing/>
  <existing>hello,</existing>
  <stat>1/1</stat>
</root>
```

The translation is found in the translation field. Notice the rest of the fields, if a word can't be translated it will be indicated, also every translated set of words (Frengly decides how to split these) can be found individually in translationFramed. That's a nice option to detect incorrect translations.

Limit of use

Reading the ToS[51] there isn't any limitation about the usage, except the systematic crawling of the service, that is not this case.

Issues

The wait time between queries must be 3 seconds, this is a serious problem if two instances are running at the same time. A solution for one instance is waiting 3 seconds every each call to ensure the next

[51] http://www.frengly.com/#!/privacy

can be performed without this problem.

The commented problem with the default XML format for errors.

Hablaa

Overview

Founded in 2011, Hablaa provides a simple translation page[52] with a complementary paid service provided by human translators[53]. Most of the content comes from the Hablaa's translators and other translation sites like Linguee[54], but users also can add and vote translations.

API

The API[55] supports translations, translations with examples and translations for similar concepts. This last feature could be useful, but after some tests, the source terms can be phrases and remote concepts, therefore this is not used for the implementation.

There isn't an API key.

Input

Endpoint	http://hablaa.com/hs/translation/
HTTP method	GET
Content type	text/html
Parameters	text_to_translate: Text or word for translation lang_code_src: Source language code lang_code_dst: Target language code

Table 3.18: Hablaa input

There aren't exactly parameters, as is showed in the next curl query, the input data is directly part of the URL separated by slashes.

[52] http://hablaa.com/
[53] http://hablaa.com/order-translation/
[54] http://www.linguee.com/
[55] http://hablaa.com/api/

Curl

In this curl query is showed how must be put the parameters:

```
curl -X GET 'http://hablaa.com/hs/translation/mouse/eng-spa/'
```

Output

The response is a JSON object like this:


```
[
   {
      "text":"!‘Hola",
      "pos":{
         "code": None,
         "title": None
      },
      "source":"hablaa"
   },
   {
      "text":"hola! saludo",
      "pos":{
         "code": None,
         "title": None
      },
      "source":"Hablaa.com"
   },
   {
      "text":"!‘hola",
      "pos":{
         "code": None,
         "title": None
      },
```

```
      "source":"Hablaa.com"
   }
]
```

Limit of use

From the ToS: "API is free to use, regarding indicated data source license. There is a limit of call that may be done from one IP in fixed period of time, to prevent from abuse. The limit is not strict, there is heuristics that guesses whether queries comes from robot or human. If there are too many queries or they look non-human – IP gets blocked. If you are a developer and such case happens: please contact us."

Issues

The language codes must be the specified in this[56] JSON object.
The implementation of this service was difficult because it works with CloudFlare and pyCurl is not allowed, that is why is used httplib2.

Glosbe

Overview

Glosbe is a collaborative project that involves the community to provide translations and also some sites like Wiktionary[57], OpenSubtitles[58] and OmegaWiki[59].
The main site offers translations between language pairs, showing an extend number of translations, with context annotations, audios, possibility of editing, google translation, similar terms and phrases in the original and the target languages (later we will see how is managed all this data).

[56] http://hablaa.com/hs/languages/
[57] http://www.wiktionary.org/
[58] http://www.opensubtitles.org/es
[59] http://www.omegawiki.org/

API

The documentation[60] explains the basic usage to obtain translations, but also how to add new of them. There isn't API key, the entire site is free.

Input

Endpoint	https://glosbe.com/gapi/translate?
HTTP method	GET
Content type	text/xml
Parameters	phrase: Text to be translated from: Source language code dest: Target language code format: Type format, can be XML, JSON or JSONP (set XML) pretty: Adds tabulations and returns for more readability (set false)

Table 3.19: Glosbe input

XML and JSON objects have the same value information, but the JSON format has less information about the keys, and these are the unique way found to obtain valid translations, so the format must be XML.

Curl

A curl query has the next appearance:

```
curl -X GET 'https://glosbe.com/gapi/translate?
from=eng&dest=spa&format=xml&phrase=hello&pretty=true'
```

Output

The output of this translator is extremely large (1125 lines), only are showed the relevant parts. As explained in the input section, the XML format has more information, and this is essential in order to obtain the translations.

The basic structure is:

[60] `https://glosbe.com/a-api`

```
<?xml version="1.0" encoding="UTF-8"?>
<map>
   <entry>
      <string>result</string>
      <string>ok</string>
   </entry>
   <entry>
      <string>authors</string>
      <map>Information about authors</map>
   </entry>
   <entry>
      <string>dest</string>
      <string>es</string>
   </entry>
   <entry>
      <string>phrase</string>
      <string>hello</string>
   </entry>
   <entry>
      <string>tuc</string>
      <map>List of meanings, translations and examples</map>
   </entry>
   <entry>
      <string>from</string>
      <string>en</string>
   </entry>
</map>
```

Inside the tuc field there is a complex and irregular structure that combines different information, most of them useless. The valid translations are found inside the structures with key com.google.common.collect.RegularImmutableMap:

```
<com.google.common.collect.RegularImmutableMap resolves-to="[...]$SerializedForm">
   <keys>
      <string>text</string>
      <string>language</string>
   </keys>
   <values>
      <string>hola</string>
      <string>es</string>
   </values>
</com.google.common.collect.RegularImmutableMap>
```

The JSON object doesn't has this key, so it's more difficult to find this structure.
Finally the translation is located inside the first string of values.

Limit of use

From ToS: "API is free to use, regarding indicated data source license. There is a limit of call that may be done from one IP in fixed period of time, to prevent from abuse. The limit is not strict, there is heuristics that guesses whether queries comes from robot or human. If there are too many queries or they look non-human – IP gets blocked. If you are a developer and such case happens: please contact us." Notice these ToS are exactly the same as Hablaa.

Issues

Some time spent searching a logic pattern to obtain the useful content from the large XML documents. Also the decision between XML or JSON was well studied.
The length of the returned data is too large, for some reason the callback function of the default pycurl function fails and only the final part of the XML can be stored, that is why is used the httplib2 library.

iTranslate4

Overview

In association with other translation services, iTranslate4 probably has the most complete machine translation capabilities in terms of alternative sources. The external supported translators are: Amebis[61], Apertium[62], Linguatec[63], Lingenio[64], Morphologic[65], Promt[66], pwn.pl[67], SkyCode[68], Systran[69] and Trident[70]. Most of them are paid services that work as desktop applications, for this project they can't be used. Also the languages of the webpages don't help to find information about these.

API

The service supports a set of dictionaries that help to decide the context of the text to obtain better translations. During the configuration of the service after the registration, for every language pair the user can choose which translators are used and their relevance.
The documentation[71] has all the information about how it works, including the response code, the encoding, the available functions and the output depending of the used input format.
The API key can be generated in the settings section[72] after the registration. In this same page can be found the number of remaining characters to translate using the current account.

Input

The configuration to call this service is the next:

There are more parameters like rid to use another configuration of translators and frm to decide the format of the input text (currently only text and HTML).

[61] http://presis.amebis.si/
[62] https://www.apertium.org/
[63] http://www.linguatec.de/
[64] http://www.lingenio.de/
[65] http://www.webforditas.hu/
[66] http://www.promt.com/
[67] http://translatica.pl/
[68] http://webtrance.skycode.com/online.asp?current=6&setlang=en
[69] http://www.systransoft.com/
[70] http://www.translate.ua/
[71] http://itranslate4.eu/en/api/docs
[72] http://itranslate4.eu/en/api/settings

Endpoint	http://itranslate4.eu/api/Translate?
HTTP method	GET
Content type	text/json
Parameters	dat: Text to be translated src: Source language code trg: Target language code auth: Private key dom: Context of the input text (general by default) min: Minimum number of translations max: Maximum number of translations

Table 3.20: iTranslate4 input

Output

The response is a JSON object like this:


```
{
   "dat":[
      {
         "text":[
            "hola"
         ],
         "length":4,
         "rid":"ape.ts"
      },
      {
         "text":[
            "Hola"
         ],
         "length":4,
         "rid":"lgt.ts"
      },
      {
         "text":[
            "!‘Hola"
         ],
```

```
            "length":5,
            "rid":"pro.ts"
        }
    ]
}
```


The translations are found in the fields text. The field rid indicates which translator is used (in this case ape is Apertium, lgt is Linguatec and pro is Promt).

Limit of use

10.000 output translated characters, that's all, this services is only a demo of the paid service, after exhausting the quota, the API key is useless.

Issues

If the minimum and the maximum number of translations aren't the same, as it has been tested, the number of translation is the minimum value, so maximum could be a big number and the minimum the number of wanted translations.

Dict.cc

Overview

Created by Paul Hemetsberger in 2002, Dict.cc[73] is a collaborative project with an initial set of translations from Beolingus's dictionary[74] and Mr Honey's Business Dictionary[75]. Users can upload their translations in a complete form, and others can correct and rate these.
The entire site is focused in the German language, so it's logically the best translations are in this language. Also the webpage has an English-German training vocabulary to learn new words between these languages.

[73] http://www.dict.cc/
[74] http://dict.tu-chemnitz.de/
[75] http://www.mrhoney.de/

API

This translator doesn't has an API, but the user rbaron[76] from GitHub created a Python library[77] that obtains the translations downloading the entire HTML document. It can be cloned from GitHub[78] or installed using pip: `pip install dict.cc.py` The library also works with command line but the implementation uses the core to obtain the translations, that is, the dictcc.py file.

Input

Endpoint	http://<subdomain>.dict.cc/?
HTTP method	GET
Content type	text/html
Parameters	subdomain: <source_language><targer_language> s: Text to be translated

Table 3.21: Dict.cc input

The user agent used is Mozilla/5.0 (Windows NT 6.3; WOW64; rv:30.0) Gecko/20100101 Firefox/30.0.
The subdomain is the concatenation of the two language code like this:

`http://enes.dict.cc/?s=computer`

Where en is English and es is Spanish.

Output

The wrapper returns a Dict object that contains the source and target languages, the number of translations and a list of tuples with source terms and their translation. This list of tuples not only contains the original term, it also has similar terms derived of this:

```
[('computer', 'ordenador'), ('computer-aided', 'asistido por ordenador'), ('car computer',
'ordenador de a bordo'), ('laptop computer', 'ordenador portátil'), ('personal computer',
'ordenador personal')]
```

The implementation is able to choose all the translations or only where the source term is exactly the provided, depending if the parameter exactly_term of the constructor is true or false, by default is

[76] https://github.com/rbaron
[77] https://github.com/rbaron/dict.cc.py
[78] https://github.com/rbaron/dict.cc.py.git

true.

Limit of use

There isn't a specific limit, further, the use of the wrapper anonymises who are doing the requests. In the FAQ[79] can be found the next: "All the linking and integration methods mentioned here can be used for free and without having to ask for permission. Maintainers of websites receiving more than a few million page views per month should however please notify me in advance to avoid possible performance issues (paul at dict.cc). Short notes from other website maintainers (after integration) are also very welcome, just because it's interesting to see which sites link to dict.cc. Server-side processing of any kind is not allowed, currently there is no API available or planned."

Issues

The wrapper solves a problem with the order of the elements of the tuples, the implementation needs to manage this order to return the correct data.
The day the creators of this site modify the position of the translations the wrapper will stop working.

Baidu

Overview

As Yandex, Baidu is the Chinese Google, it has around 42 products, most of them search engines (text, music, news, statistics) and thematic services like Baidu Space, Baidu Encyclopedia and Baidu Library.
The translator page[80] can translate text form a specific language pairs and how this is pronounced. Aside from the translator, Baidu also provides a dictionary[81] in English and Chinese.

[79] http://www.dict.cc/?s=about%3Afaq
[80] http://translate.baidu.com/
[81] http://dict.baidu.com/

API

The documentation[82] is in Chinese, so it's recommendable an automatic web translator to try to understand it.

The API supports an array of source texts separated by return characters, but for this implementation only is done for an individual translation.

Following this tutorial[83] the registration process is more easy. It is required the API key, but also exists the client key and the secret key.

Input

Endpoint	http://openapi.baidu.com/public/2.0/bmt/translate?
HTTP method	GET
Content type	application/json
Parameters	q: Text to be translated from: Source language code to: Target language code client_id: API key obtained after registration

Table 3.22: Baidu input

Don't confuse the client_id parameter with the client key.

Output

The response is a JSON object:

```
{"to": "spa",
"from": "en",
"trans_result": [{"src": "hello", "dst": "Hola"}]}
```

The translation is found in the dst field of the array trans_result.

[82] http://goo.gl/dodzCT

[83] http://wp-autopost.org/manual/how-to-apply-baidu-translator-api-key/

Limit of use

From the documentation and after an English translation is found: "Baidu translation api currently divided into four files, for ordinary developers to provide 1000 cycles/hour limit"

Issues

Some problems to obtain the verification code in the registration process.
The language of the site is Chinese.

Discarted translators

The next translators were tested for their integration, but these finally cannot are available for various reasons.

Apertium

In collaboration with Spanish and Romanian institutions, this translator[84] currently works with Python 3, running a server (Toro and Tornado) in the local machine and doing petitions to localhost. The technologies used are incompatibles with the project, that uses Python 2.

Watson

It is the translator[85] of IBM integrated in the Bluemix[86] platform, the services only offers 30 free days, also, a registered and deployed application must be done following the guidelines, this fact is quite far of the required services.

[84] `https://www.apertium.org/`
[85] `http://www.ibm.com/smarterplanet/us/en/ibmwatson/developercloud/machine-translation.html`
[86] `https://console.ng.bluemix.net/`

MyGengo

The philosophy of MyGengo is to upload jobs, check if these are completed, and when are done, get the results. The services is focused in human translation, but also exists a sandbox scenario with a machine translator, but this last works equally, it is not received an automatic response. There is a library[87] for Python but it is useless for this project.

BabelFish

Searching the origins of Bing Translator it is found BabelFish[88], but sadly there isn't an API that works with this translator.

WorldReference

Currently this service[89] doesn't provide new API keys.

Other online translators

Internet is full of translator pages and they only are accessible using a browser, as for example dict.cc. The translations only can be obtained downloading the entire HTML document and try to parse it. Some sites have methods to detect this practice, so first the wrapper can be banned, and second, site can change the display of the elements as commented in the dict.cc translator.

3.3 NewsdeskTranslations

3.3.1 Introduction

As explained in previous sections, NewsdeskTranslations is a Django application in development where the users can obtain semantic data related to plant pests from a remote database using SPARQL

[87] `https://pypi.python.org/pypi/mygengo/1.3.1`
[88] `http://www.babelfish.com/`
[89] `http://www.wordreference.com/`

queries, for later validate it and sent the new derived valid terms to the original database using natural language techniques, translation services and other semantic databases.
In addition to the integration of MultiTranslator in NewsdeskTranslations, it require some adjustments and new features related with the generation and validation of the obtained data.
First, we will take a look at the requirements of the application, then which type of data manages the application, after the visible part of the project, where the users will be working, and finally, how all these functionalities are implemented.

3.3.2 Requirements

At the end of the development this application must accomplishes the next requirements:

- The user must be able to use a webpage to store terms using natural language tools to obtain derivations, translators to obtain the same terms in other languages and semantic databases to obtain additional information about the pests.
- The user must be able to choose which information about the selected pests would obtained using the webpage. For enrich, the user can choose the resources, for translate the user can choose the resources, the target languages and the translators.
- The user must be able to see a progress bar to know the current number of completed tasks in the webpage.
- The user must be able to use the Django administration panel to filter, sort, classify and search labels.
- The user must be able to export and import CSV files with labels.
- The user must be able to validate the labels inside the exported files.
- The user must be able to configure the score (weight) of each source present in the database.

3.3.3 Ontology

All the data managed by the application is obtained using SPARQL queries, that is, these information is located into a Semantic Web database.
To understand more easily the type of words used by the application and their relations, the figure 3.5 shows the used ontology.

Using this ontology, the application requests and obtains terms classified in:

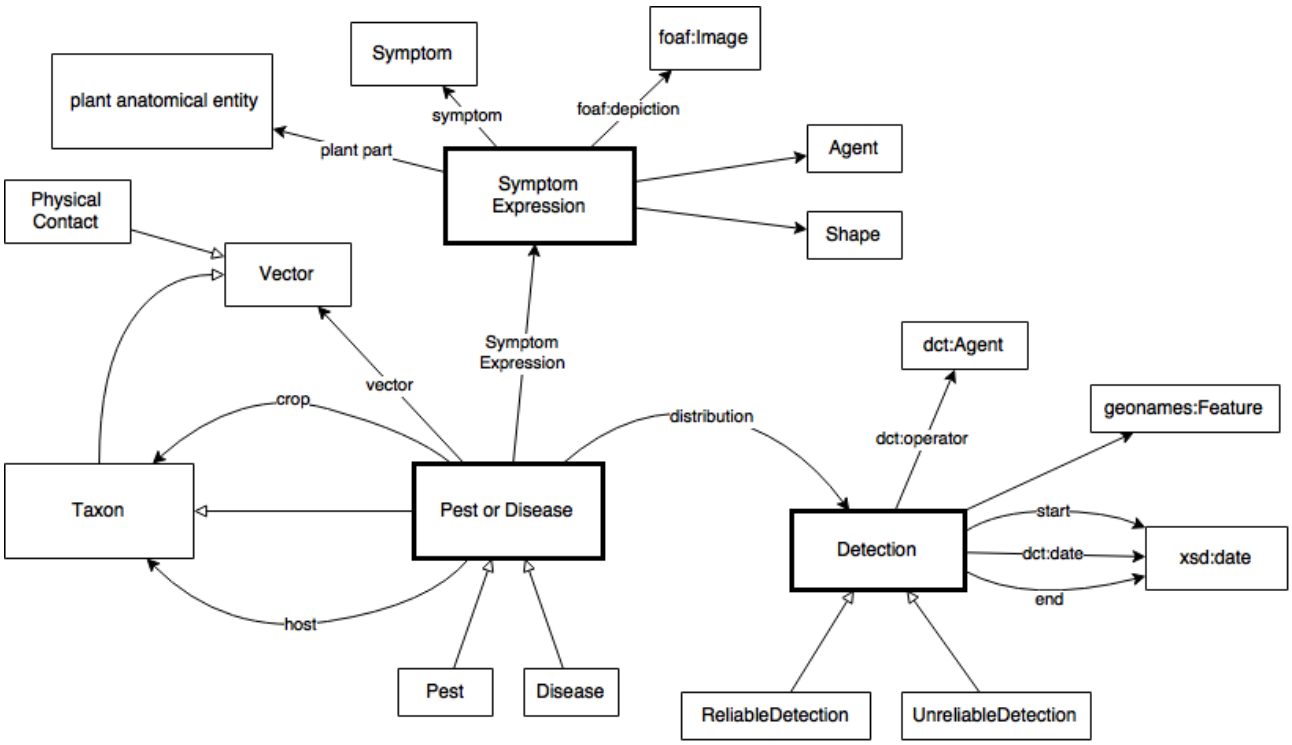


Figure 3.5: Plant pest ontology

- Plant part: The different sections that compose the anatomy of a plant, like crown, trunk, leaves and root.
- Symptom: Those symptoms that are shown by those hosts affected by the pest or disease. For example, yellowing, withering and change color.
- Crop: Specie plant affected by a pest or disease.
- Vector: Organism that helps transmitting the pest or disease.
- Transmission mechanism: Non-biological agents that can facilitate the transmission of the pest or disease. For example, mud, wind and water.

3.3.4 Front-end

The front-end is the set of user interfaces available in the client side, this project has two main pages and some auxiliary pages. The mainpage is the site where the users will generate the new labels for the required pests using some implemented tools and features. The other important page is the admin panel provided by Django, in this case reinvented as a validation panel where the labels are checked. The rest of the pages are used by the administrators of the site or for minor tasks that will be explained.

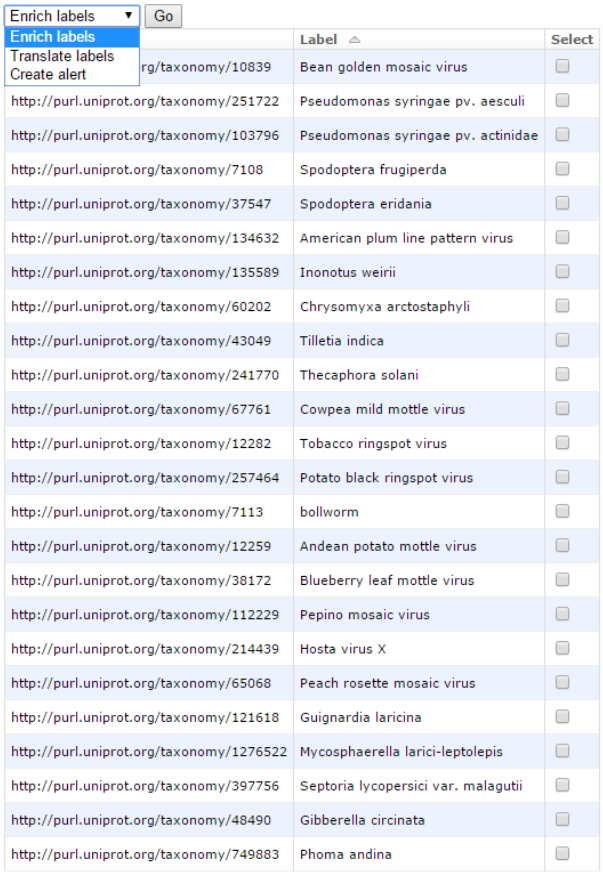


Figure 3.6: Original website

Webpage

When I started the project, the webpage has the basic aspect showed in the figure 3.6.

After lot of changes and tests the figure 3.7 shows the proposed design and utilities to generate new labels from pests.

Actions

At the left the user can use the droplist to enrich labels from the semantic web database, DBPedia and EPPO and translate labels, the make alert option is a future feature doesn't contemplated in the project. When the action is decided only remains to click the "Go" button to start the execution. Also there is the "Pending tasks" button to check if the tasks are completed, showing the progress bar section, that is because the tasks are independent of the client, that can close the tab or browser and return after few minutes to see the new progress, the unique restriction is that the user must use

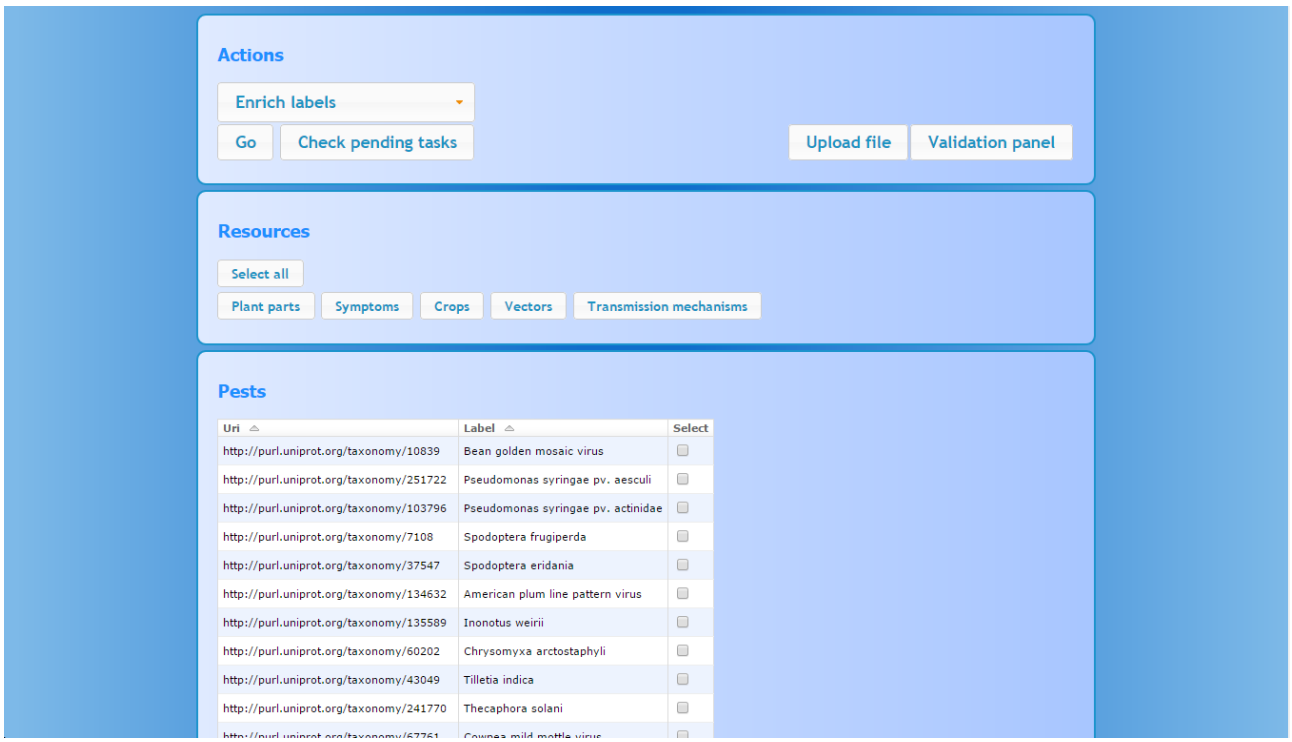


Figure 3.7: New website design

the same computer and browser to enter again in the webpage.

At the right there are the "Upload file" and "Validation panel" buttons, with the first, the user can upload a CSV file with labels (explained in the section 3.3.4), and with the second he goes to the administration panel to manage the generated labels.

Pest table

The pest table is the original table implemented by Josep Maria, any change was done. When a user interacts with the website, this is the first election that he must done, because if he changes the table page, depending of the browser, the checked options are deleted.

Options

Depending of the chosen action, more or less options will be available.

For the enrich labels action it is showed the resources section. For enrich with DBPedia and EPPO all the options are hidden because the unique required data are the pests. Finally, the translate labels option has all the sections: resources, languages and translators, as it can be seen in the next image.

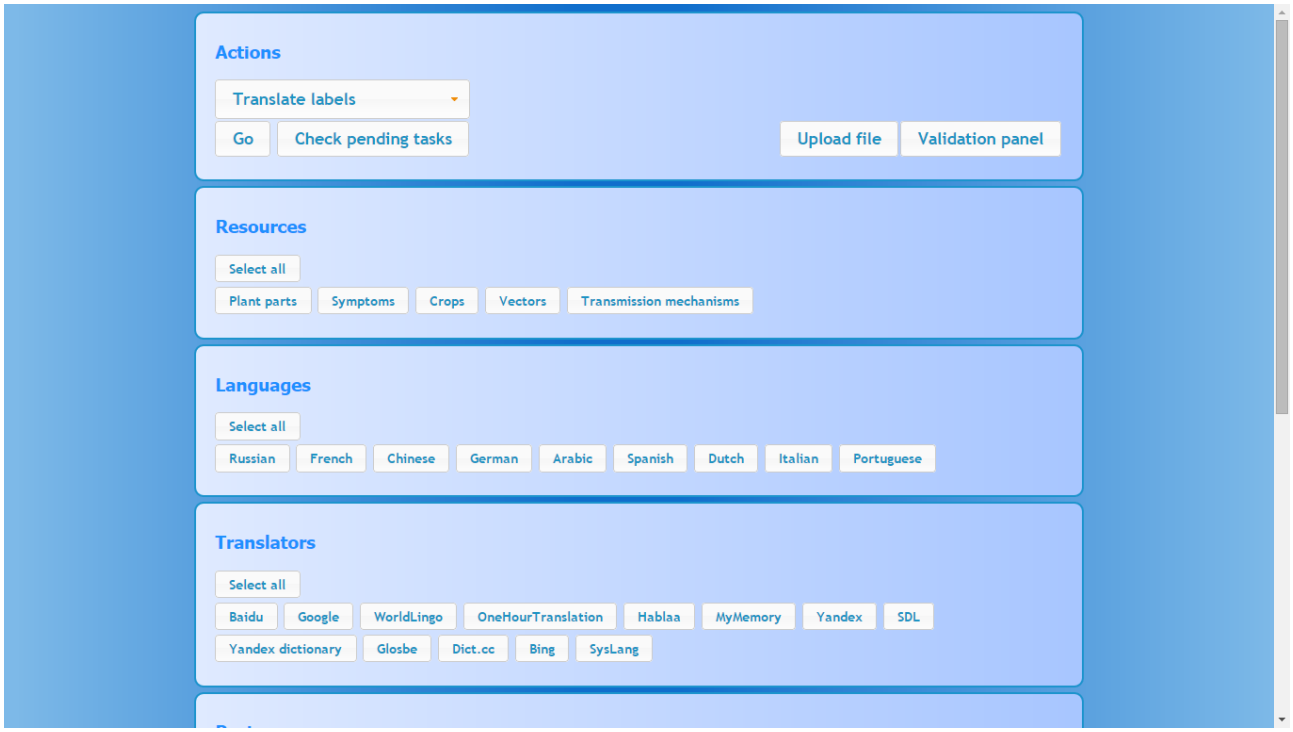


Figure 3.8: Translate sections

Progress bar

When an action is executed, the user must wait the completion of the tasks, otherwise, unexpected results will be obtained. As explained before, the user can close the browser and check after the state with the "Check pending tasks" button. The first step is to obtain the labels selected in the resources section and the desired pests. An animation is showed because the server doesn't know the number of labels.

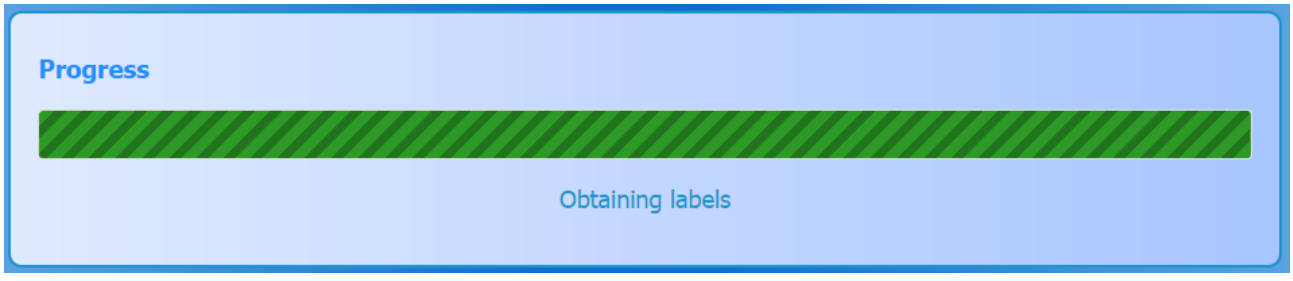


Figure 3.9: Progress bar waiting

When the labels are obtained, the progress bar will advance to fill. At the bottom of the bar will be displayed the percentage, number of completed tasks at the current time and total tasks.

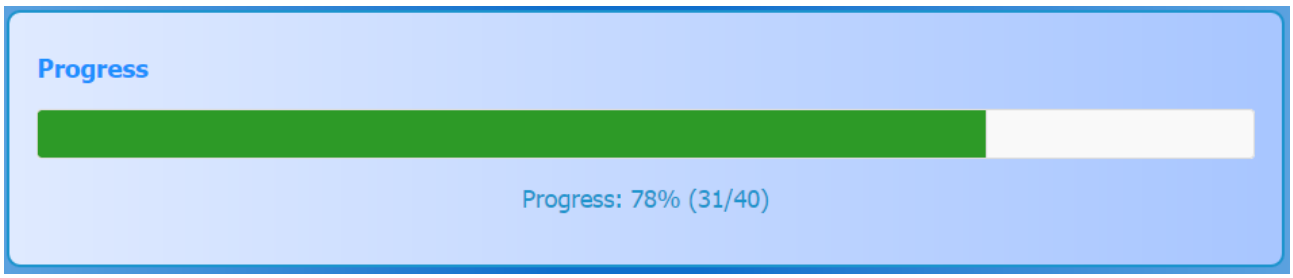


Figure 3.10: Progress bar advancing

Finally the tasks are completed, a message is showed indicating which action is completed, if the user uses the "Pending tasks" button this message won't be displayed.

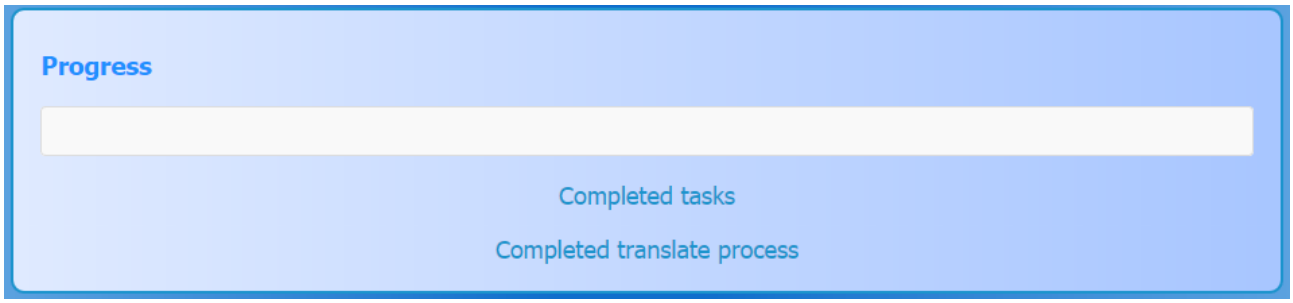


Figure 3.11: Progress bar advancing

Admin panel

The admin panel is a default webpage provided by the Django framework to manage the models of the database. In this case the users must use this panel to manage the generated labels in the mainpage and additional features that will be explained. In the root of the admin webpage there are the available pages:

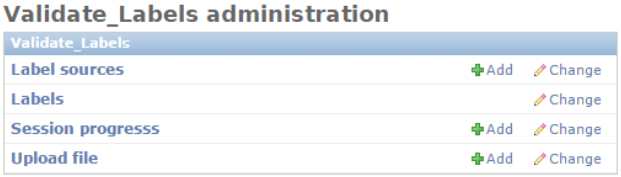


Figure 3.12: Models administration

Validation panel

In this page the users can manage the labels generated in the mainpage.

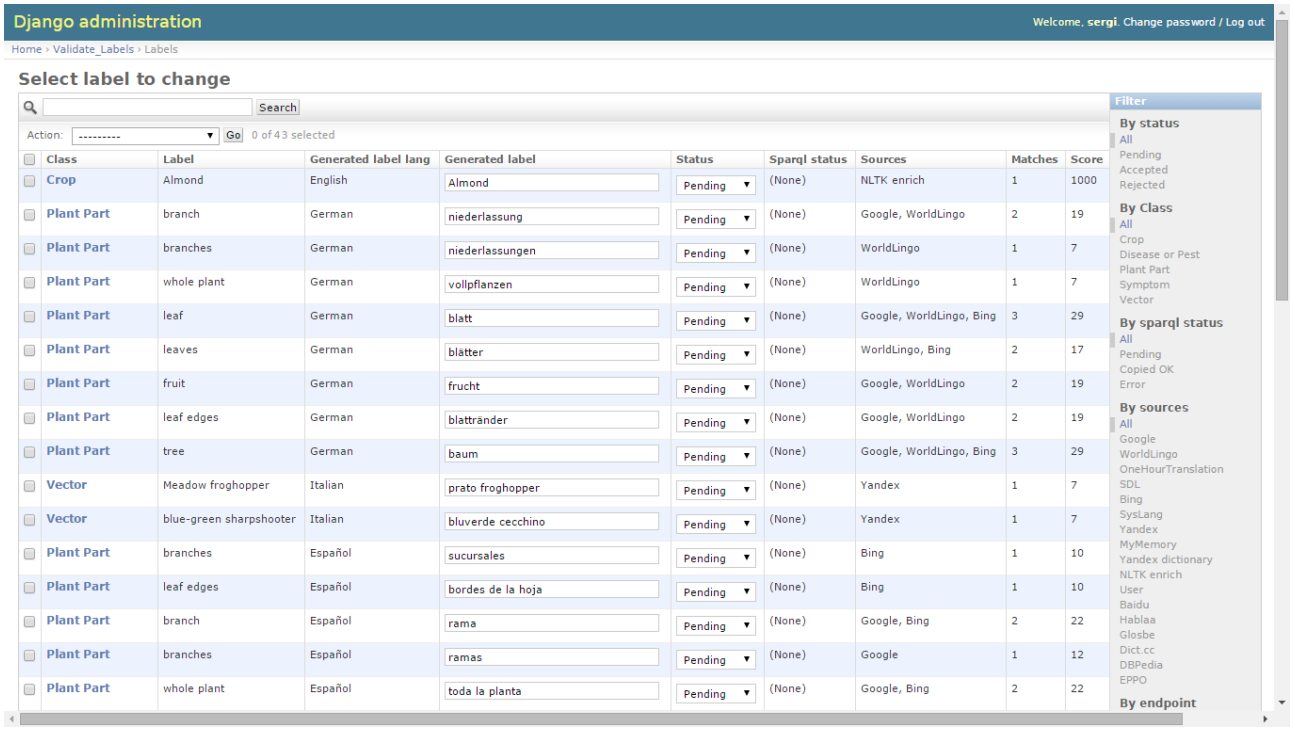


Figure 3.13: Models administration

The main functionality is accept or reject labels using the status column and the save button at the bottom of the page. Initially all the labels are in the "Pending" state, user can decide which are good, checking the "Accepted" state, or bad, checking the "Rejected" state. When all is decided and saved, the labels are sent to the remote database.

At the top there is a search bar to find labels using the columns label and generated label.

Another important feature are the filters, it's extremely recommended the use of these at least to filter the "Pending" labels from the rest.

The filters used filter by:

- Status
- Class
- SPARQL status
- Source
- Endpoint

- Language

Finally, after the use of the search bar and the filters, the last feature to manage the labels are the sorted columns, except the sources column, the headers of the rest columns can be clicked to sort the labels ascendant or descendant by the values of the selected column.

Custom actions

Django provides a droplist to implement new actions.

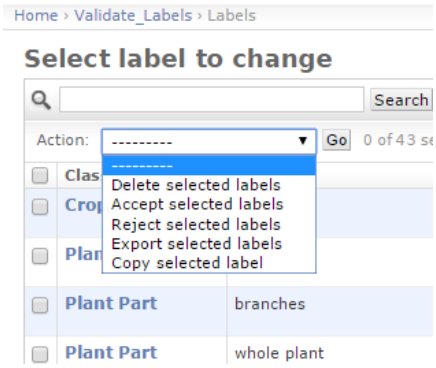


Figure 3.14: Custom actions

Accept, reject and delete labels

Complementing the default validation process, the user can use the action droplist to accept, reject or delete all the marked labels using the left checkboxes.

Copy a label

If user needs, the duplication of labels is available, but only one each time to avoid problems, for example, cloning 100 labels at the same time could be a chaos, it is better to go one by one, cloning, changing and accepting a label, and repeat the process for all the desired labels.

Export labels

If the user doesn't want to use the validation panel, he can use a CSV file with the marked labels. The export action launches a dialog to decide where the file is saved, it can be showed in the figure 3.15.

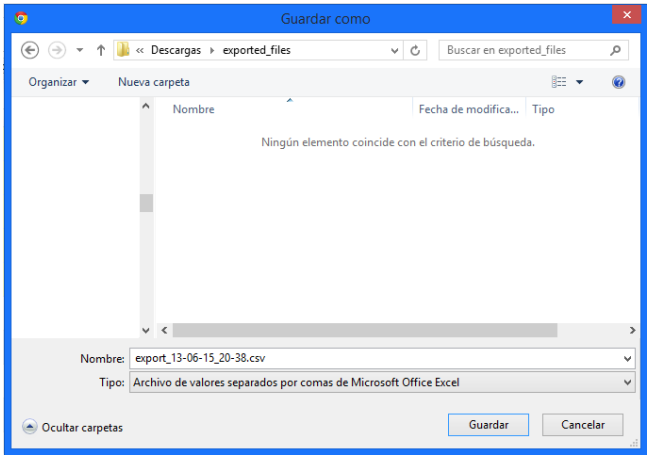


Figure 3.15: Export file dialog

	A	B	C	D	E	F
1	Version	1				
2	id	URI	Original term	Target language	Generated label	Sources
3	6067	http://purl.uniprot.org/taxonomy/3755	Almond	English	Almond	NLTK enrich
4	6066	http://www.cropontology.org/rdf/PO:0025073	branch	German	niederlassung	Google, WorldLingo
5	6065	http://www.cropontology.org/rdf/PO:0025073	branches	German	niederlassungen	WorldLingo
6	6063	http://www.cropontology.org/rdf/PO:0025034	leaf	German	blatt	Google, WorldLingo, Bing
7	6064	http://www.cropontology.org/rdf/PO:0000003	whole plant	German	vollpflanzen	WorldLingo

Figure 3.16: Exported file

Obtaining a file like the showed in the figure 3.16.

The unique column that users can edit is the Translation column, the others must be ignored. The unique rules to verify the labels are if the text is deleted, the label is rejected, else, it is accepted. To manage the labels, can be activated some sort and filter features that the spreadsheets include.
The version number is included if in future updates the proposed format changes, avoiding problems with old files, that will continue using the current implementation.

Upload file

After the verification of an exported file, the user can use the "Upload button" in the mainpage and send the file using the provided dialog showed in the figure 3.17. The labels will be read and directly classified and updated.

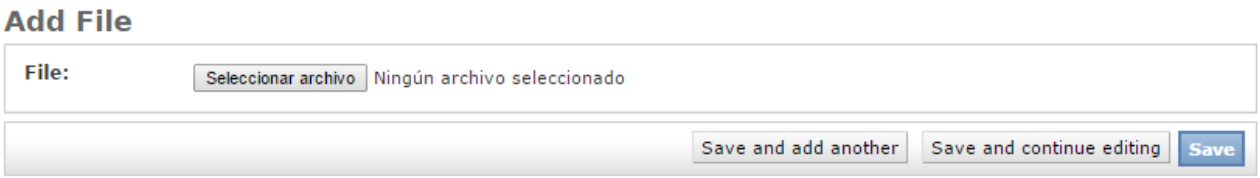


Figure 3.17: Upload file page

Edit labels

This functionality only must be used to solve issues like to correct the language of a label, or to check the information of the label, and exists the possibility to broke the label, for example, changing the URI.

Manage resources

In the last column of the validation panel there are the scores for each label, they are the sum of the score for each source, this value is used to sort the labels with a better criterion than the number of sources, as long as these scores are well-balanced.

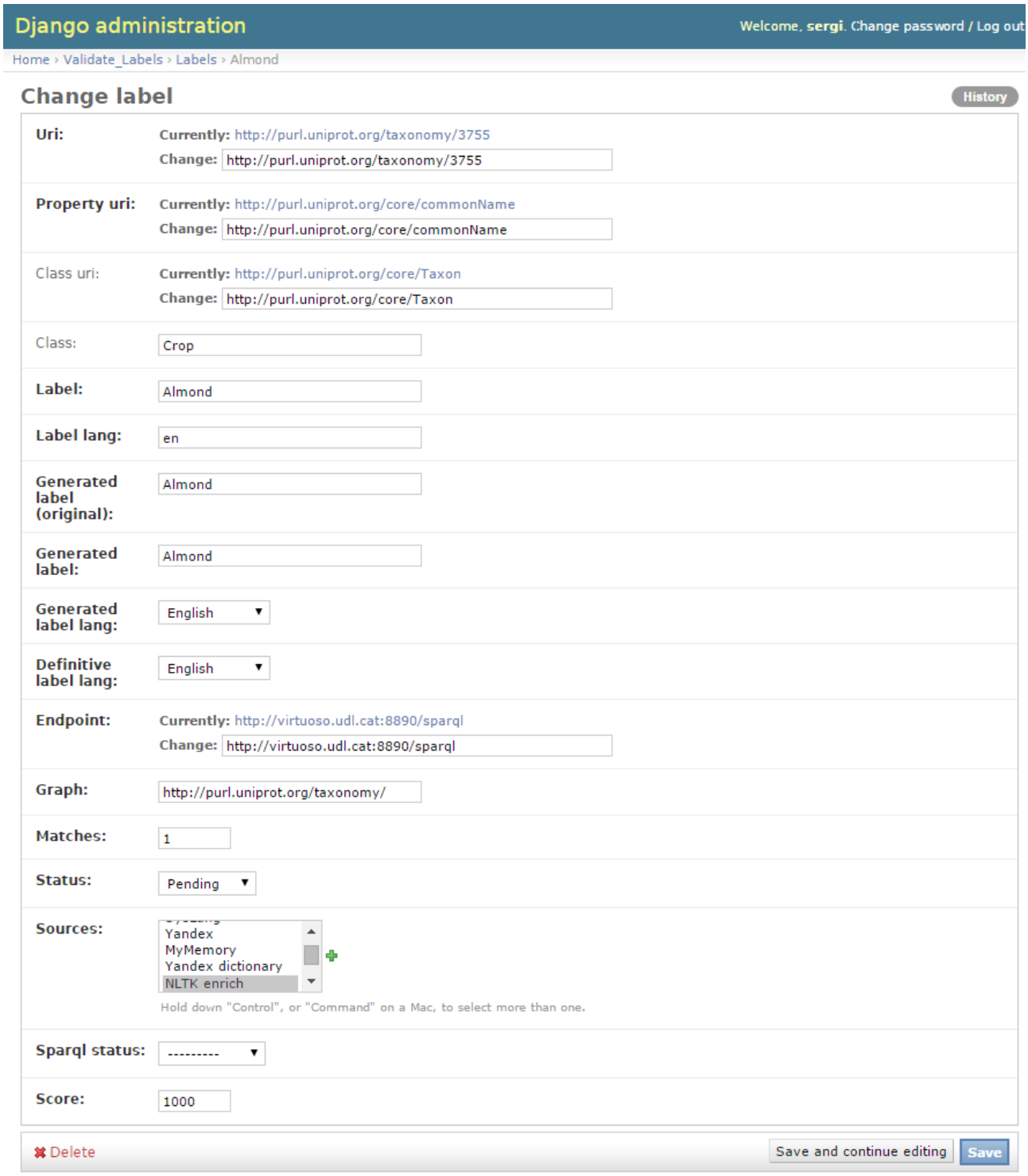


Figure 3.18: Edit label page

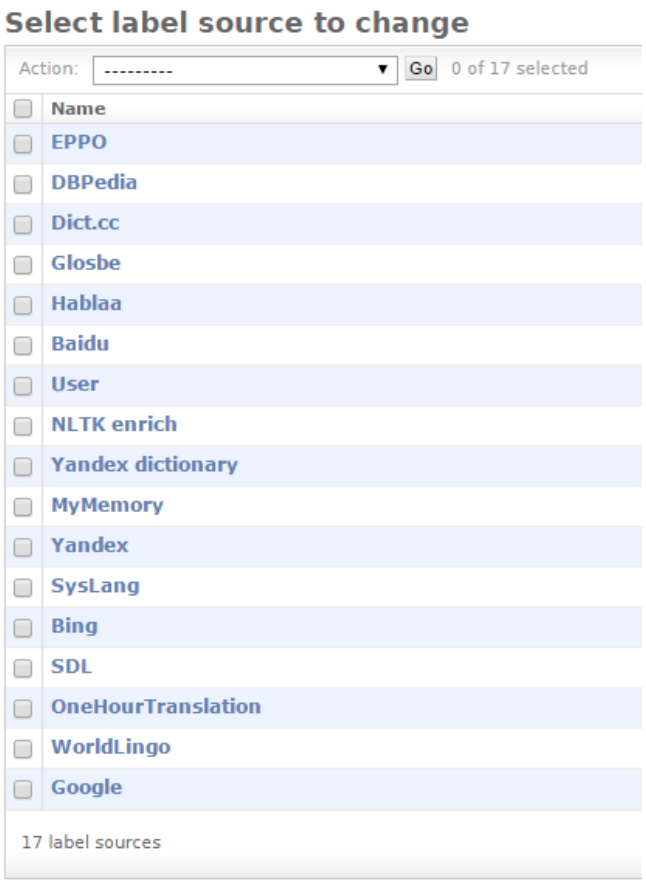


Figure 3.19: Sources page

These scores must be chosen by the administrators of the site when a new source appears. After the modification of a source, all the labels that has this source are recalculated. Notice any source can be deleted to prevent inconsistency in the scores of the labels.

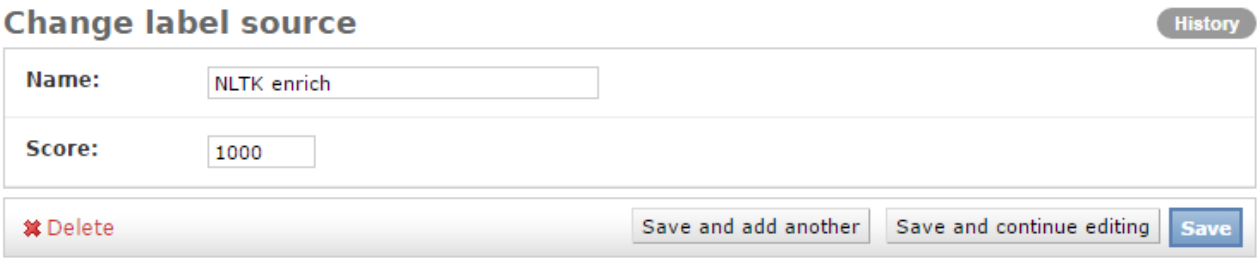


Figure 3.20: Score update page

Sessions

Currently, this page is only for debugging, a normal user doesn't need it. This is explained in the section 3.3.5

3.3.5 Back-end

The back-end is the server side, where the Django application is running and serving the pages requested by the client side.

Webpage

Initially the web page only contains a droplist with the actions enrich, translate and make alert. The form and the enrich process worked, also the translate process using goslate (Google). As it can be seen before, the page has changed a lot.

Actions

Enrich

The enrich process first obtains all the required labels from the remote database using the SPARQL queries, after that the enrichment process really starts, depending of the structure of the labels, some derivations are done using NLTK:

- Verbs: Derivation of the original verb form to obtain the infinitive, gerund and particle and the lexema.
- Nouns: Plural and lexema of the word.

Also a treatment about punctuation symbols, extra spaces and lowercase characters is done.

DBPedia and EPPO

The DBPedia and EPPO actions were implemented by Roberto, my job was to connect these with the rest of the actions, adding these in the views file and sending it the correct parameters to these

functions.

The DBPedia enrichment launches a SPQRQL query to the DBPedia endpoint, parsing and obtaining the scientific names of the selected pests.

The EPPO enrichment first obtains the EPPO code from the ontology and a query is done using the API provided by the EPPO service[90], like translation services, after the data is obtained, it is parsed and the new labels for the desired pests are obtained.

Translate

This is the time to remember the MultiTranslator project, this module, and most specifically Transfusion, was created to encapsulate the logic behind the translator services and mainly offering a simple function (get_translation) that does all the work.

During the development, transfusion acts as a git submodule, offering me an easy way to update the repository and apply the changes directly in the Django project.

At the end of the development, MultiTranslator became an independent project, that is, every change must require an update from the repository, but in this case this fact doesn't affect, because only Transfusion is used. The keys and settings for Transfusion were directly wrote in the Django project, for now, all the translators except iTranslate4 are available for the users, probably over time some translators will be excluded, for example Hablaa, that becomes unavailable during some time if different users execute it at the same time, or Syslang, that needs a 3-second delay. And probably, others will be added as they will be appearing or being available without fee.

The basic behavior of the translate function is:

1. Set up the Transfusion instance with the translators desired by the user.
2. Receive the labels that will be translated, together with the language required.
3. Execute the get_translation function and obtain the Translation instance after the execution of the corrector.
4. Parse the Translation instance received and insert the translations into the database, taking into account which translator is used every time.
5. Update the progress bar at the end of every individual task.

[90] `https://data.eppo.int/`

Pest table

When the webpage is loaded, the server launch a query to obtain all the available pests in the remote database, serving the entire webpage and rendering the table with the pests.

Options

The different sections required for each action were directly implemented in the HTML code, using Django templates to generate the entire sets of buttons. Every group has an identifier for the "Select all" buttons.

Progress bar

Decisions

When the translate process were working, a big issue appears, if the user only wants few translations, for example, the crops in Spanish using Google for a unique pest, the process is faster, a few seconds, but if a big amount of data is required (like the first practical case of the Grouper), the time taken is extremely big, it could be more than 20 minutes requiring all the sources, languages and translators. So an additional task is proposed an accepted, the implementation of a progress bar, very used in file services like DropBox.

Initially I started a research process to know how must be implemented a progress bar, one of the bests options found was Celery, a Django application that manages asynchronous tasks, and this project[91], at first appearance seems to be easy to use, but the fact is that only the process to install all the required components (MySQL for the database and RabbitMQ for the message passing, among others) and interconnect these was very difficult for me. After some days expended on this, finally I desisted.

The second option was the combination of JQuery-UI or Bootstrap for the design, Ajax for the asynchronous communication with the server and JavaScript/JQuery to update the progress bar. But it stills remaining an issue, how the server knows which user is requesting the new progress bar value? The answer is using a new model combined with the session token that Django provides for each specific computer and browser.

[91] `http://www.dangtrinh.com/2013/07/django-celery-display-progress-bar-of.html`

Implementation

The new model, called SessionProgress, stores the session_key of a user, the number of completed tasks and the total number of tasks. When a user request the mainpage, a SessionProgress instance is created in the database, storing the session_key and initializing the tasks to -1. Also, initially the first time the mainpage is served, the progress bar values are set to -1 by default, that is, the first time the Ajax communication hasn't begun. It's important to comment that the HTML code for the progress bar is in an independent file, linked in the mainpage.
At the client side, an Ajax call requests the progress bar template every 500 milliseconds, when the server receive the petition, it obtains the session of the user and sends the renderized template with the new values to the user. Now the client renders again the progress bar section with the current and total tasks, depending of the values, the progress bar has a different aspect.
If total tasks and current tasks are greater than 0, the user executed an action and is waiting, the progress bar displays the current state of the execution. Else, if current progress is 0, the labels aren't obtained yet, the animation is showed. Finally if the two previous cases don't occur, the progress bar is void because all the tasks are completed.
Every action available in the droplist updates the progress value every time an individual task is completed, that is for example, if a user enrich a pest, at the end of every NLTK enrich, the progress is updated. In the case of the translations, the total tasks are the multiplication of the pests times number of labels times languages times translators.

Design and user interaction

Initially there isn't temporalized the dessign because my knowledge about web design and front-end development was very poor and it could affect at the temporalization of other tasks, and I say it in past because in the middle of the development (with good temporalization dates) I started to learn JQuery, CSS, JavaScript and Ajax to improve my skills and offer a better first impression of the project, because except the Django panel, there isn't any other graphical user interface.
As a disclaimer, the development of this part were hard and some of the code is taken from examples and tutorials, like any other newbie web developer, so probably an expert on this field would do better the logic of the functionalities and know more libraries than me.
In general, most of the JavaScript code uses JQuery 1.11.2 and the graphics were done with JQuery-UI

1.11.4. Also there is a library for Ajax using JQuery 1.11.1 and another for the forms, adapted by Malsup.
The next list shows how functions are used to implement every element:

- Texts: Simply changing the CSS colors and sizes. Simply changing the CSS colors and sizes.
- Buttons design: Using the button() function from JQuery and overriding the default CSS colors.
- Pest table: The pest table is the unique element from the original webpage, it uses django-tables2 for the design and the form.
- Sections: Using individual divs for each section and modifying the border and border-radius parameters.
- Select all buttons logic: Getting the elements of the section where is the button and changing the checked parameter for each button.
- Sections update using the droplist: Using the slideUp and slipdeDown functions.
- Progress bar design: Using the provided progress bar design provided by JQuery.
- Progress bar automatic update: Using the setInterval("refresh()", 500) function to update the bar every 500 milliseconds.
- Gradients: Using the CSSmatic[92] webpage.

Admin panel

As explained before, the admin panel had the capacity to validate labels and sent it to the remote database using the SPARQL queries, all this part were done at the start of the project and my work continue right it left off.

Refactoring

In the temporalization can be seen a high quantity of deletions, that is because during the firsts weeks, and also when it is required, the original code is highly modified to offer more generic solutions and doing the necessary adaptations when new functionalities were implemented.
The refactor process affect at the next parts.

[92] `http://www.cssmatic.com/`

SPARQL queries

All the queries to obtain the different classes had a duplication of the code because in some cases the URI parameter is required and in others not, creating the same query with and without this parameter, an easy solution was found, in the place where the URI was wrote, a new function decides if the URI must be wrote or not depending if it is None or not.

Database insertions

Initially there were lots of functions to obtain the different resources of a pest, the structure of these were equal, only the parameters of the SPARQL query changed, so a big important reimplementation were done to offer a more generic solution, including the compatibility with the translation function. The first step was the creation of a new class, LabelInfo, to store in memory all the labels obtained, and also, the creation of a set of enumeration classes to manage the used actions, graphs and resources. This is the dictionary corresponding to the plant part resource, it can be seen how is configured every parameter to be used later in the enrich and translate processes:

```
plant_part = {  "common":   {"resource_type": "plantPart",
                             "class_uri": "http://purl.obolibrary.org/obo/PO_0025131",
                             "class_label": "Plant Part"},
                "local":    {"endpoint_type": settings.ENDPOINT,
                             "graph_type": settings.GRAPH,
                             "property_uri": "http://www.w3.org/2000/01/rdf-schema#label",
                             "function": sparql.get_local_plant_parts},
                "remote":   {"endpoint_type": settings.REMOTE_ENDPOINT,
                             "graph_type": settings.GRAPH_PLANT_ONTOLOGY,
                             "property_uri": "http://www.w3.org/2000/01/rdf-schema#label",
                             "function": sparql.get_remote_plant_parts},
                "enrich":   {"verb_ing": False}
            }
```

After this refactor, a new function was implemented using these data structures, replacing the old duplicated functions, and the function that inserts the data into the database also was reimplemented. Finally, the enrich and the translate functions also were remade with this new behavior.

Custom actions

Accept, reject and delete selected labels

The delete action is a default action provided by Django.
The functions that implement the accept or reject actions receive which labels are checked, changing the status depending of the case, and finally saving all.

Import/export file

The export functionality reads all the checked labels and store it using the unicodecsv library, that is because the default csv library provided by Python can't manage unicode strings. The file is wrote as presented in the front-end section and sent to the client as an attachment:

```
response = HttpResponse(content_type='text/csv')
response['Content-Disposition'] = "Attachment; filename=export_%s.csv" % \
(time.strftime("%d-%m-%y_%H-%M"))
```

Copy

This functionality clones the selected label, changes the generated label name appending `"_copy_<index>"`, where index is the number of copy, finally, the source User is assigned to this. The user can clone the same label the number of times he want, for example obtaining: insects_copy_1, insects_copy_2, insects_copy_3 and so on. If the user tries to clone more than a label, a Django error message will be showed at the top of the validation panel.

Minor features

The next additions are little improvements that weren't present in the original implementation.

Search bar

Adding the next line in the admin file the search bar was activated:

```
search_fields = ['label', 'definitive']
```

Score column

The field score was added in the LabelSource and ValidateLabel models, and the score column was activated for the validation panel. As explained in the front-end section, when the score of a source is updated a post save receiver is triggered and it updates all the related labels.

Languages

Internally, the server works with language codes, like Transfusion, but for the users it is better to display the name, not the code. Using a dictionary where the key is the code and the value is the name, Django automatically works with the codes but shows the names.

3.3.6 Practical case

With the next example, we will across the full set of functionalities of the project, indirectly including the MultiTranslator module. We take the case that a new pest is detected in the lands of Lleida, the pest is called Xylella Fastidiosa, a bacterium that affects the olive and almond trees (among other trees), this bacterium is found in some vector insects of the Cicadellidae family, when the insects feed the wood of the tree, if these tree has the bacterium, the insect will transport it to another trees when it feed again. The disease of Xylella Fastidiosa is the withering and death of the affected tree.

Figure 3.21: Insect from the Cicadellidae family

Figure 3.22: Affected olive tree with Xylella Fastidiosa

Previously, in the remote database a investigator must provide some guidelines and information about this pest, classifying this in crops, plant parts, diseases, the common names he know and so on.

Enrich

Mainpage

The investigator responsible of the NewsdeskTranslations project receive the order to generate the necessary labels to use these in the new filter to obtain the news about this pest.
He goes to the mainpage and does the next actions:

1. Finds Xylella Fastidiosa in the pest table and check it.
2. Checks the "Select all" button for the enrich labels action.
3. Presses the "Go" button to start the enrichment and waits.

Validation panel

When all the tasks are completed he presses the "Validation panel" button and goes to the Django admin panel, where all the labels are generated, he filters these by "Pending" and "NLTK enrich" to obtain only the desired labels.
Now the investigator must decide which generated labels are correct and which are rejected, or change these if he knows the correct derivations of the words. When all the pending labels are updated and sent to the remote database, this initial process is completed.

Enrich with DBPedia/EPPO

Mainpage

Now the investigator wants to know which are the alternative names of the pest, he uses the DBPedia and the EPPO enrichments.

Validation panel

Filtering by "Pending" and DBPedia or EPPO he can validate these new labels.

Translate

Mainpage

In 2013 this pest was extended by Italy, so besides the Spanish, the Italian could be a good option to obtain more information. The investigator choose the "translate labels" option, select all the translators and the languages Spanish and Italian, and finally proceeds to start the translation process using the "Go" button.

Validation panel

Our investigator is Spanish, so he can validate the Spanish labels, but he doesn't know Italian, so first he validate the Spanish words using the correct filters. He can be helped by the columns "Matches" and "Scores" if he is doubting between two words, for example. Also, if he notices that a important translation is not present, he can clone it using the copy action and put his own translation. He has some contacts that know Italian, so he exports the Italian labels and sent the generated file to one of his co-workers. Some days ago the file is validated and returned to the investigator, the last step is the upload of the file, all the labels will be updated, saved and sent to the remote database automatically.

Conclusions

After this example, the investigators who will use this tool may have an idea of how it works and what are the available functionalities. It also can be used as a feedback to obtain new ideas or changes that are reclaimed if a feature is not good as expected.

Chapter 4

Conclusion

4.1 Future Work

4.1.1 MultiTranslator

Lots of new features can be implemented, new translators and writers, better corrector implementations, new scripts and functionalities about unexploited features, like the interconnection of the current tools, or the dump of data to write again it in other formats. Probably the most important feature is the implementation of more source languages. For this project only English is required, but for a more general purpose, new languages could be activated after the required tests and fixes for each translator.

Some translators can be configured to filter translations by user, type, quality, and so on, a deeper analysis can be done to improve the possibilities of these features.

4.1.2 NewsdesksTranslations

This project is still work in progress, with many months remaining till its deadline. For now, the generated data requires human intervention before it is loaded into the semantic repository. One of the ways to facilitate this process could be the automation of the validation of translations using a score for their reliavility. To determine this score, a significative sample of translators output should be validated by experts and contrasted with what is generated by the MultiTranslator.

The Semantic Web is evolving every day, if we wait a little bit more, new semantic databases that

integrate and enrich existing datasets can help us to obtain verified translations and the type of terms. Also new ontologies and specific databases could be added besides EPPO and DBPedia.
Currently the users can't know if the used remote services are working correctly, additional information like the number of new labels or status of the translators, SPARQL endpoints and databases could be displayed when the tasks of the user are completed, to inform the administrators if there are problems. The "make alert" action in the droplist of the mainpage must be implemented when the application that gets the validated labels for the filters is finished.

4.2 Conclusions

The investigations done in this project provided me a deeper knowledge about Semantic Web, machine translators and to lesser extent, natural language processing. The fully integration of all the studied exceed the available time to develop the project, that is, focusing the efforts only in MultiTranslator was a good idea. If it had tried to implement all some essential features couldn't been implemented, for example, if Transfusion hadn't the corrector implementation, the quality of the translations would be poor, and therefore, the entire project wouldn't obtain the minimum quality to be a useful tool.
The design and development of MultiTranslator show me how a Python programmer really must work and which problems must face. At the beginning my knowledge about dependencies and installation process were basic, but after the last the project became one more of the thousands libraries created by the community to solve specific tasks, and that facilitate and speed up the development of new projects, like the set of libraries used in this project, without these some of the features couldn't have been implemented. Also, the fact that MultiTranslator is available for everyone is a good starting point for other developers that require translation services in their applications, NewsdeskTranslations will be useful for a lot of people, but its existence will have fewer impact in the development community.
The addition of new features in NewsdeskTranslations notice me the importance that Django is obtaining during the last years, it not only provides a set of tools and abstractions to develop powerful web applications, but these tools are well designed and easy to integrate and extend, as it can be seen in the project.
Another important task studied and finally implemented with good results is the web design, this type of programming never attracted me, now I have a better understanding of these technologies and after crossed the first barriers about syntax, paradigms and the different methodologies to work, all that remains is apply all the acquired knowledge and skills in new projects.

Finally, the tools and functionalities implemented reach their final stage for this project, but it has to keep working both in this project and the posterior and subsequent projects related to this.

Bibliography

[1] Apertium api. http://wiki.apertium.org/wiki/Apertium-apy.

[2] Collins dictionary api. http://www.collinsdictionary.com/api/.

[3] First steps with celery: how to not trip. http://hairycode.org/2013/07/23/first-steps-with-celery-how-to-not-trip/.

[4] First steps with django - using celery with django. http://docs.celeryproject.org/en/latest/django/first-steps-with-django.html.

[5] Gengo api docs. https://github.com/gengo/gengo_api_docs/blob/master/content/v2/api_methods/service.md.

[6] Google translate api documentation. https://cloud.google.com/translate/v2/libraries.

[7] Language translation with python. https://impythonist.wordpress.com/2014/02/11/language-translation-with-python/.

[8] Linked data client: Modules. http://marmotta.apache.org/ldclient/modules.html.

[9] Machine translation tools. http://wiki.crisiscommons.eu/wiki/Machine_Translation_Tools#APIs.

[10] Memsource machine translation. http://wiki.memsource.com/wiki/Machine_Translation.

[11] Nltk - language processing and python. http://www.nltk.org/book/ch01.html.

[12] Nodebox - loading the library. https://www.nodebox.net/code/index.php/Linguistics#loading_the_library.

[13] Openlogos machine translation. http://logos-os.dfki.de/.

[14] Python - parallelizing cpu-bound tasks with multiprocessing. http://eli.thegreenplace.net/2012/01/16/python-parallelizing-cpu-bound-tasks-with-multiprocessing.

[15] A ruby wrapper for the wordreference api. https://github.com/malandrina/word_reference.

[16] Smartling translation api. https://docs.smartling.com/display/docs/Smartling+Translation+API.

[17] Using watson machine translation service with watson explorer. https://github.com/Watson-Explorer/wex-wdc-integration-samples/blob/master/wex-mt/watson-machine-translation-readme.md.

[18] What is machine translation? http://www.systransoft.com/systran/corporate-profile/translation-technology/what-is-machine-translation/.

[19] Wordreference api. http://www.wordreference.com/docs/api.aspx.

[20] Wordreference workflow. https://github.com/mbdw/WordReference.

[21] J.M. Brunetti, R. García, R. Gil, and A. Granollers. Development and testing of the media monitoring tool medisys for the monitoring, early identification and reporting of existing and emerging plant health threats. Technical report, Universitat de Lleida, 2015.

[22] Des. La evolución, expansión, diversificación y extinción de los idiomas. https://elimperiodedes.wordpress.com/2013/03/29/la-evolucion-expansion-diversificacion-y-extincion-de-los-idiomas/.

[23] Aurelia Drummer. Literature review: Machine translation.

[24] Roberto García. Introducción a la web 3.0.

[25] Roberto García. Marcado semántico en la web 3.0.

[26] Paul Houle. How to write sparql queries against freebase data. http://blog.databaseanimals.com/how-to-write-sparql-queries-against-freebase-data.

[27] Stefan Kottwitz. *LaTeX beginner's guide*. Packt Publishing Ltd, 2011.

[28] Paul Miller. Demonstrating the value of sparql to the semantic web. http://www.zdnet.com/article/demonstrating-the-value-of-sparql-to-the-semantic-web/.

[29] Antonio López Muzás. Sistema modular de presentación de información para la plataforma de web semántica rhizomer. Master's thesis, Universitat de Lleida, 2009.

[30] ReportLab. "reportlab pdf library. Technical report, ReportLab, 2014.

[31] Toby Segaran, Colin Evans, and Jamie Taylor. *Programming the semantic web.* " O'Reilly Media, Inc.", 2009.

[32] WorldLingo. Service api to the worldlingo system. Technical report, WorldLingo, 2014.

[33] Patrick Sinclair Libby Miller Stephen Betts Yves Raimond, Tom Scott and Frances McNamara. Case study: Use of semantic web technologies on the bbc web sites. `http://www.w3.org/2001/sw/sweo/public/UseCases/BBC/`.