WikipediaBase

Chris Perivolaropoulos

Sunday 21 February 2016

Contents

1 Introduction

WikipediaBase base is a backend to START responsible for providing access to wikipedia related information. The WikipediaBase we refer to is a python rewrite of the now deprecated Ruby WikipediaBase.

2 People

The python implementation was initially written by Chris Perivolaropoulos and was eventually handed over to

- Alvaro Morales
- Michael Silver

3 Functionality

The WikipediaBase server provides operations to:

- find all classes that a Wikipedia term (object) belongs to, and
- given a specific wikipedia term and a class under which it functions, provide information

3.1 Queries

3.2 Ontology

1. Terms
2. Classes

3.3 Infoboxes

Infoboxes are tables displaying information about a wikipedia page in a semi structured way. At the moment most of the data retrieved by WikipadiaBase comes from infoboxes

{{TODO: example image}}

4 Getting started

4.1 Language

The WikipediaBase implementation that we refer to is written in python. Previous implementations were written in Java and Ruby but the language of choice for the rewrite was python for multiple reasons:

- Python is in the pre-graduate curriculum of MIT computer science department. This will ease the learning curve of new members of Infolab.
- Python is a easy to learn and mature language with a rich and evolving ecosystem. This fact eases the introduction of new maintainers even further.

4.2 Getting the code

The entire WikipediaBase resides in a git repository in infolab's github orginization page

```
git clone git@github.com:infolab-csail/WikipediaBase
```

4.3 Virtualenv and dependencies

WikipediaBase depends on multiple other python packages. Fortunately, python is shipped not only with a great package manager, but also with a mechanism called virtualenv that isolates installations of a project's dependencies from the rest of the system, thus avoiding problems like version or namespace collisions. The way this effectively works is that the global python installation is half copied half symlinked to a local directory and the dependencies are installed only in the local sandbox. To create and activate a python virtualenv:

```
$ virtualenv --no-site-packages py
$ . py/bin/activate
$ which python
/the/local/directory/py/bin/python
```

Now that we can safely install anything we want without breaking any global installation

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

We will need some extra stuff for WikipediaBase to work:

- Postresql
- Redis

The installation of these packages varies across platforms. Both these packages are databases. Their purpose is for caching and for storing ahead-of-time compitations like infobox markup name to rendered name matching.

4.4 Backend databases or live data

5 Architecture

5.1 Pipeline

When resolving a query WikipediaBase employs a pipeline of modules to figure out what the best way to respond would be.

1. Frontend

   WikipediaBase can be used as a library but it's primary function is as a backend to START. The communication between START and

WikipediaBase is carried out over a plaintext telnet connection on port {port} using EDN-like sexpressions. The frontend handles the network connection with START, translates the received queries into calls to knowledgebase and then translate the knowledgebase response into properly formulated sexpressions that it sends back over the telnet connection.

(a) Protocol

2. Knowledgebase

   The knowledgebase is the entry point to the rest of wikipediabase. It uses the Provider/Acquirer pattern to transaprently provide the frontend with arbitrary methods. Those methods are responsible for chosing whether we are to resort to classifiers or resolvers (or any other mechanism) for answering the query. Available classifiers and resolvers become accessible to the knowledgebase automatically using their base class.

3. Classifiers

   Each classifier is a singleton that implements a heuristic for assigning a class of an object. Thereare a couple classifiers available at the moment.

4. Resolvers

   Resolvers are also singletons but their purpose is to find the value of the requested property.

5. Lisp types

   Lisp type instances are wrappers for python objects or values that are presentable in s-expression form that START can understand. They are created either from the raw received query and unwrapped to be useful to the pipeline, or by the answer WikipediaBase comes up with and then encoded into a string sent over telnet to START.

5.2 Fetcher

The fetcher is an abstraction over the communicatioln of WikipediaBase with the outside world. It is a singleton object that implements a specific interface.

5.3 Infobox

5.4 Article

6 Provider/Acquirer model

WikipediaBase attempts to be modular and extendible. To accomplish this it is often useful to have parts of the system that access resources (eg. heuristic methods) without knowledge of what module those came from. Additionally it is often the case that resources come from many different modules. To avoid ad-hoc code and hard dependencies the provider / acqirer model was created:

- Subclass provider/Acquirer classes
- The Provider uses the `@provide` decorator to provide resources.
- The acquirer has transparent access to the aggregate of provided values for a key.

7 Testing

7.1 Unit testing

The good functioning of WikipediaBase is assured by a comprehensive test suite of unit tests, functional tests and regression tests.

1. Unit tests

   Unit tests test small blocks of functionality, that are composed to create the system at large. For unit testing we use python's default testing library. Each test is a class the subclasses

2. Functional and regression tests

   Functional tests are tests written before, during or shortly after the development of a system and they assert the correct overall functioning of the system. Regression tests are very akin to functional tests. They prove that a found bug was fixed and assert that it will not appear again later. Functional and regression tests currently reside in `tests/examples.py`

7.2 Examples

8 Synonyms

8.1 Good/Bad synonyms

8.2 Synonym generation

9 Backend databases

9.1 DBM

9.2 SQLite

9.3 Redis

9.4 Postgres

10 Data sources

11 Date parser

Dateparser resides in a separate package called overlay-parse

11.1 Parsing with overlays

The concept of an overlay was inspired by emacs overlays. They are objects that specify the behavior of a subset of a text, by assigning properties to it. An overlay over a text t in our context is tuple of the range within that text, a set of tags that define semantic sets that the said substring is a member of, and arbitrary information (of type A) that the underlying text describes. More formally:

$$o_i \in TextRanget \times Set(Tag) \times A$$
$$Text \to \{o_1, o_2, ..., o_n\}$$

So for example out of the text

$$The weather today, \overbrace{Tuesday}^{o_1}\ \overbrace{21^{st}}^{o_2}\ of\ \overbrace{November}^{o_3}\ \overbrace{2016}^{o_4},\ was\ sunny.$$

We can extract overlays $\{o_1, ..., o_4\}$, so that

$$
\begin{array}{lll}
o_1 = (& r(\text{"}Tuesday\text{"}), & \{\text{DayOfWeek}, \text{FullName}\}, \quad 2) \\
o_2 = (& r(\text{"}21^{st}\text{"}), & \{\text{DayOfMonth}, \text{Numeric}\}, \quad 21) \\
o_3 = (& r(\text{"}November\text{"}), & \{\text{Month}, \text{FullName}\}, \quad 11) \\
o_4 = (& r(\text{"2016"}), & \{\text{Year}, \text{4digit}\}, \quad 2016)
\end{array}
$$

Notice how for all overlays of the example we have $A = \mathbb{N}$, as we encode day of the week, day of the month, month and year as natural numbers. We encode more precise type information (ie that a day is inherently different than a month) in the tag set.

Once we have a set of overlays we can define overlay sequences as overlays whose ranges are consecutive, that is their and their tag sets match particular patterns. For example we can search for sequences of overlays that match the pattern

$$p = \text{DayOfMonth}, \text{Separator}(/), (\text{Month} \wedge \text{Number}), \text{Separator}(/), \text{Year}$$

to match patterns like 22/07/1991, where $Separator(/)$ matches only the character "/"

11.2 The implementation

11.3 Optimization

1. Comparison

11.4 The dates example

11.5 Benchmarks

12 Future

12.1 Configuration

1. Persistence
2. Pass by reference
3. Lenses
4. Laziness
   (a) Referential (Ref - Items)
   (b) Computational