Designing and developing text processing tool (**Scalpel**)

Andrei Vishneuski, Jun 2012

Why ?

- Introduce software design which:
  - has simple input and output data (token, text, etc)
  - establishes clear correlation between written code and scientific problem the code is supposed to deal
  - provides rich set of text processing operations as number of elementary, easy to use modules and classes
- Integrate various third party solutions under “common roof” covered by unified API
- Minimize required installation and configuration activities

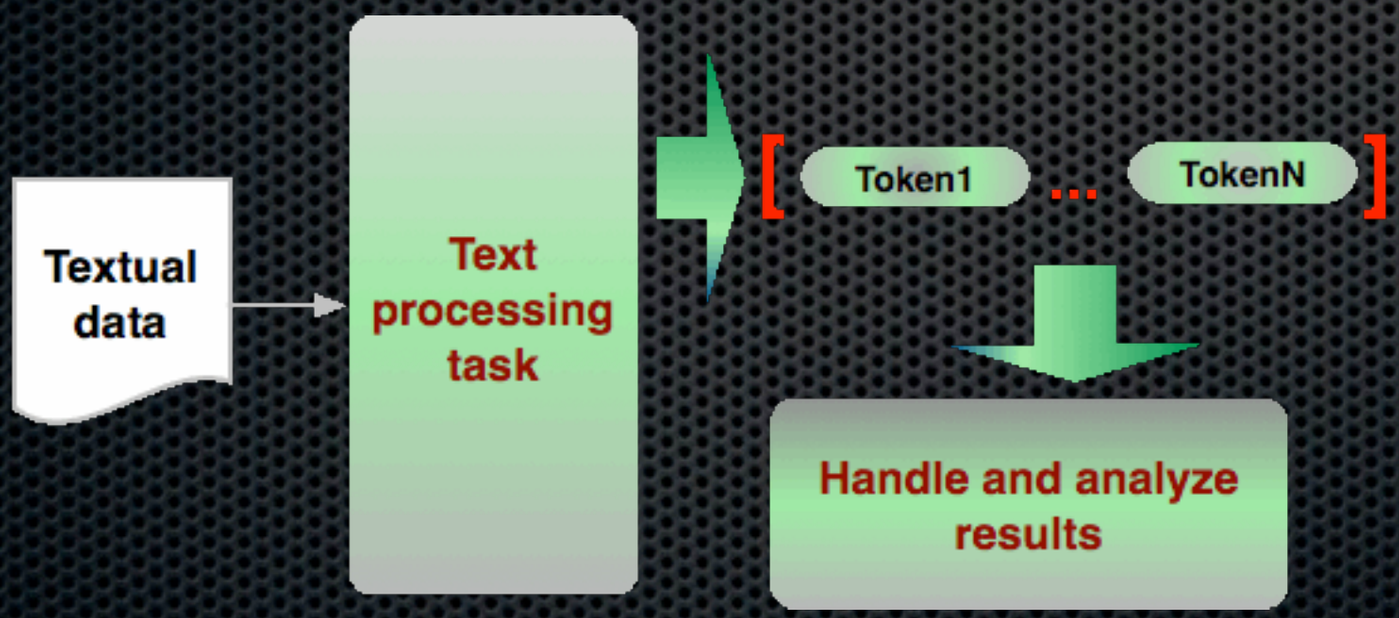
Design: Few principles

- Widely Python callable feature - construct and call class:
  - *TextProcessingTask([args1, args2, ... argsN])([text, ...])*
- Prefer using “yielding” to allocating and returning a list
  - *[t* **for** *t* **in** *CoNLL2002.testa().tokens()]*
- Weak modules dependencies to avoid “all depends on all” mess. Specific concept should not be referenced by common concept.
- Minimalistic well unified output and input data. Token tuple is key structure:
  - *FuzzySearch(max_errors=3)(text, pattern) -> token*
  - *POS()(tokens) -> [token, ...]*

Design:text processing abstraction

In many cases text processing tasks can be viewed as a tokenization task

Token can be generalized as tuple:
(<Text>, <offset>, <length>, <type>)



- Named entities recognition
- Named entities normalization
- Stemming
- Part of speech tagging
- Sentence splitting
- etc

Design:text processing abstraction

Sample text: "**Obama** *has visited* **Moscow**. **Barak Obama** *has been invited by government of* **Russian Federation**."

Tokenization results for different text processing tasks:

- NER: ("Obama", 0, 5, PER), ("Moscow", 18, 6, LOC), ("Barak Obama", 26, 11, PER) , ("Russian Federation", 50,18, LOC)
- NEN: ("Barak Obama", 0, 5, PER), ("Barak Obama", 26, 11, PER), ...
- POS: ("visited", 10, 7, VERB), ...
- Stemming: ('visit', 10, 7, None), ...
- Sentence splitting: (None, 0, 25, SENT), (None, 28, 68, SENT)

Design: Textual information

- Various textual sources implementations should extend “Text” class
- All text sources implementation has to be representable as unicode text, no special property to keep text is expected:
  - *TextProcessingTask()(TextFile(“/home/kindergarden/test.txt”))*
  - ~~*TextProcessingTask()(TextFile(“/home/kindergarden/test.txt”).text)*~~

- Bunch of ready to use implementations is available:
  - **TextFile** - text stored in a file on disk
  - **GzipText** - text stored in a gzipped file
  - **XmlText** - text fetched from XML
  - **TextBulk** - to keep number of texts as one combined
  - **AnnotatedText** - text with meta data (annotations)

Design: Token handling routine

- **TokenSet** should be used to analyze bunch of tokens

```
# get all location entities
ts = TokenSet(NER()("Amsterdam is ..."))
print [ t for t in ts.tokens(Token.NE_LOC)]
```

- **TokenSet** analyzing can be customized by implementing appropriate custom token matcher:

```
# get named entities with undefined position
class MyTokenMatcher(TokenSet.Match):
    def match(self, token): token[1] < 0

ts = TokenSet(NER()("Amsterdam is ..."))
print [t for t in ts.tokens(MyTokenMatcher())]
```

- **TokenSet** can be matched with other token set of token array

Design: If Token is not enough

- Token is tuple: (“Amsterdam”, 10, 9, LOC)
- If the simple approach is not enough use class that is full analog of tuple but can extend token with additional properties:

```
# Token class is more structured analog of tuple
t = Token(“aaa”, 10, 3, 0)
print t.text, t.type, t.offset, t.length
print t[0], t[3], t[1], t[2]  # this line is equivalent to previous line

# token set does not see difference between number of tuples and
# number of Token class instances
TokenSet([Token(...), Token(), ...]).tokens(...)

# user defined Token class
class MyToken(Token):
    def __init__(self, text, offset, length, type, parent_token=None):
        Token.__init__(self, text, offset, length, type)
        self.parent_token = parent_token  # extend the token with a new property
```

Integration: Third party software

- Third party software
  - C Python module integration (stemmer, etc)
  - External processes integration (TNT, Stanford, Lingpipe, etc)
    - Scripts, binary executable files (TNT, conlleval)
    - Java code (Stanford, Lingpipe, etc)

- Third party software license
  - Open source:
    - GPL and LGPL
    - Apache
    - Specific (like LBJ)
    - MIT
  - ~~Commercial~~

Integration: Host third party code

How to host third party software ?

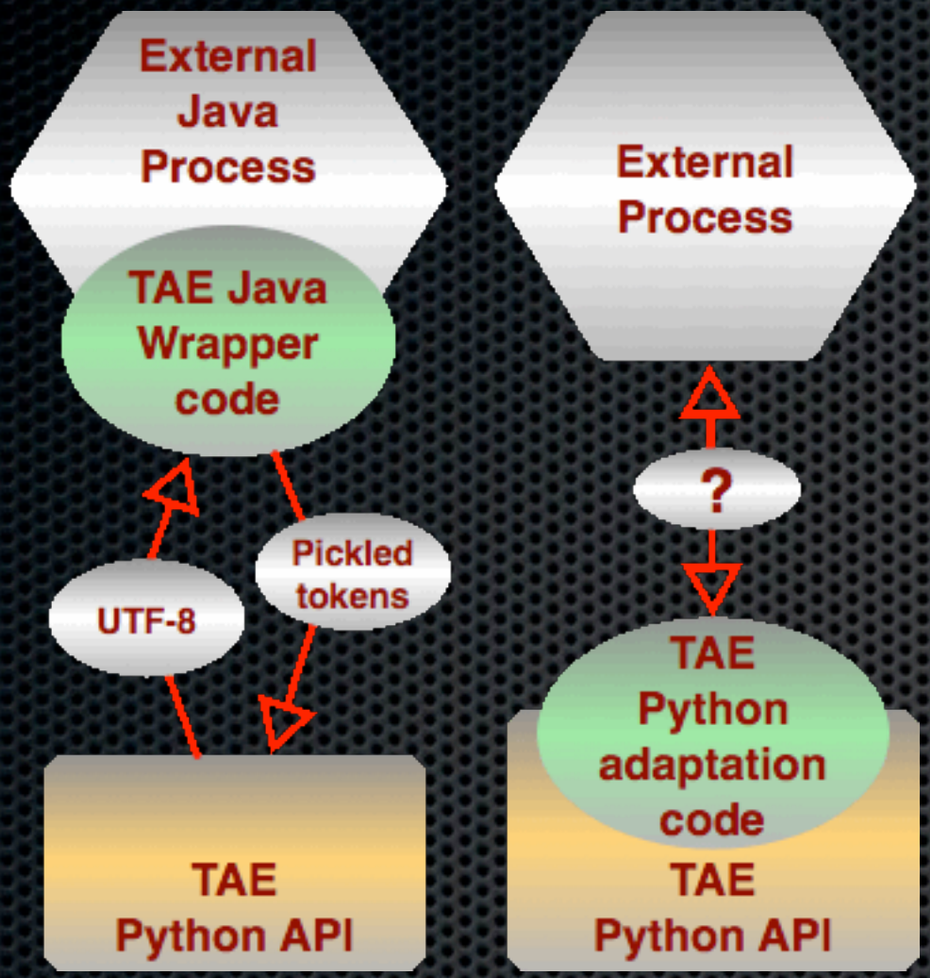
- Integrated directly into developed package
- It is up to user to install and configure it
- Keep it somewhere locally prepared for our project

Use Primus HTTP Repository:

```
<artifact repository root>
  +-- lingpipe
  |     +-- lingpipe-4.0.0.zip
  +-- stanford
  |     +-- stanford-3.1.1.zip
  +-- primus
        +-- primus-1.0.0.zip
        +-- primus-2.0.1.zip
```

- Third party package has to be placed in repository
- Primus is responsible to download and deploy it in project context
- Third party software should be kept untouchable

Integration: External processes



- Pipes are supposed to be used to communicate with an external process
- Scripts, binary executable code integration cannot be simplified by declaring a common format and number of text processing tool utility classes
- Java external processes integration is simplified by unifying exchanging protocol basing on Python pickle format as output and UTF-8 encoded data as input

Installation: Primus

- Download, run installation script and wait:
  - $ *python <tool_home>/.primus/deploy.py*
- Action the script does:
  - Validate python version and source code
  - Compile Java and C/C++ modules code
  - Download from repository and deploy third party packages in text processing tool context:
    - LBJ, Stanford NER, Stanford POS, TNT, CoNLL, etc
  - Run test cases
- Text processing tool is ready for use after Primus script completes its work

API

Core API and classes	Tokens representation Textual information representation TokenSet search, match, tokenization, distance function, etc External code integration Common code (Processes, XML, files, etc)
Named entities recognition	Stanford (english and dutch) Lingpipe (english, spanish, dutch) TNT (dutch) LBJ (english)
Part of speech tagging	Stanford TNT
Stemming	Snowball (Braun algorithm)
Corpora	CoNLL2000 (english) CoNLL2002 (dutch, spanish) CoNLL2003 (english)
Wikipedia linkage	Using our normalization DB

API

Levenshtein distance	Classical and improved implementation Python and C/C++ implementation Bunch of matrixes implementations (Python and C) Minimal path calculation (Python and C) Diagonal calculation (Python and C) Texts alignment diagonal approach (Python and C)
Bitap algorithm	Bitap fuzzy search (basing on variouse error types: insertion, deletion, substitution) and Bitap as distance function
Preprocessing	Stop word detection (dutch, english) Tokenizations etc

Sample 1: NER/POS/Stemming

Lingpipe NER

```
from gravity.tae.ner.tnt.ner import NER
tokens = NER()("Amsterdam is ...")
```

Stanford POS

```
from gravity.tae.pos.stanford.pos import POS
tokens = POS()("Amsterdam is ...")
```

TNT NER

```
from gravity.tae.ner.lingpipe.ner import NER
tokens = NER()("Amsterdam is ...")
```

List and run available POS taggers

```
from gravity.tae.pos.pos import POS
for name in POS.list(): POS.pos(name)("Amste ...")
```

TNT NER applying to text file

```
from gravity.tae.ner.lingpipe.ner import NER
from gravity.tae.text import TextFile
tokens = NER()(TextFile("/home/test/t.txt"))
```

Run TNT POS for Dutch CoNLL 2002 corpus

```
from gravity.tae.pos.tnt.pos import POS
from gravity.tae.corpora.conll import CoNLL2002
tokens = POS()(CoNLL2002.testa())
```

Snowball Stemmer

```
from gravity.tae.stemmer.snowball.stemmer import Stemmer
from gravity.tae.tokenizer import WordTokenizer
tokens = Stemmer("en")(WordTokenizer("Amsterdam is ..."))
```

Sample 2: CoNLL2002 shared

The task is evaluation a NER for the given corpus in respect of measuring recall, precession, accuracy and F1 parameters:

```
corpus = CoNLL2002.testb("nl")
print corpus.conlleval(NER()(corpus))
```

or evaluate all available recognizers:

```
for name in NER.list():
    tokens = NER.ner(name)(corpus)
    print corpus.conlleval(tokens)
```

Output:

```
processed 68875 tokens with 3941 phrases; found: 3965 phrases; correct: 2970.
accuracy:  97.34%; precision:  74.91%; recall:  75.36%; FB1:  75.13
              LOC: precision:  83.06%; recall:  84.88%; FB1:  83.96
             MISC: precision:  74.44%; recall:  59.14%; FB1:  65.92
              ORG: precision:  67.47%; recall:  69.84%; FB1:  68.64
              PER: precision:  75.49%; recall:  90.62%; FB1:  82.37
```

Sample 3: Levenshtein stuff

```
d = fLevDistance()(“abc”, “aac”)   # result is 1
m = fLevDistance().fill_matrix(DistanceMatrix('abcdd', 'aacsddd'))
print m
    000 001 002 003 004 005 006
    001 001 002 003 004 005 006
    002 002 001 002 003 004 005
    003 003 002 002 002 003 004
    004 004 003 003 002 002 003

p  = fMinPath(m)
print  m.toString('abcdd', 'aacsddd', p)
      | a   a   c   s   d   d   d
    ---------------------------------
    a |*00 001 002 003 004 005 006
    b |001 *01 002 003 004 005 006
    c |002 002 *01 *02 003 004 005
    d |003 003 002 002 *02 003 004
    d |004 004 003 003 002 *02 *03

print  fLevDistanceDiag().fillMatrix('abcdd', 'aacsddd', DistanceMarix(111))
    000 001 002 111 111 111 111
    001 001 002 003 004 111 111
    111 111 001 002 003 004 111
    111 111 111 002 002 003 004
    111 111 111 111 111 002 003
```