Balie – Baseline Information Extraction

Multilingual Information Extraction from Text with Machine Learning and Natural Language Techniques

David Nadeau

School of Information Technology & Engineering
University of Ottawa, Ottawa, Canada

January 3rd 2005

1 Introduction

This report presents Balie, a system for multilingual textual information extraction (IE). IE consist in finding and structuring data from free-written texts. Examples of IE tasks include named-entity recognition (NER), identification of proteins, abbreviation resolution, keyphrase extraction, identification of semantic roles and a large amount of much specific tasks (e.g.: finding e-mails, urls). Some tasks, like the last ones, may be easily tackled by the use of regular expressions or simple rules. However, more complex tasks like NER, require efficient and flexible architecture. Balie is driven by the need of such a flexible IE architecture.

As is, Balie will not extract a particular type of information from text but it will transform the text in a *TokenList* that is the key element in many IE tasks. The following components of Balie are required to create a rich *TokenList*:

1. Language detection;
2. Tokenization;
3. Sentence boundary detection;
4. Part-of-speech tagging;

The report is divided as follow. Section 2 presents the basic requirement of the IE task. It covers all four Balie core components on a theoretical point of view. Section 3 presents Balie on a technical point of view. It presents class diagrams, explanation of implementation choices and gives some sample code.

Balie is open source software issued under the GNU General Public License. It is hosted at SourceForge [1]. This report describes Balie v.1.02.

[1] http://balie.sourceforge.net/

2 The basics

2.1 Decision to be Multilingual

As a prelude to any information extraction journey, a crucial question has to be asked:

"Do you want your prototype / algorithm to be multilingual?"

Answer "no" and you'll be able to develop solutions for a wide range of problems with a somewhat consistent architecture and coherent class division. If, moreover, you develop a monolingual English system, you'll find many open source projects and modules like: Brill's part-of-speech tagger[2], Porter's stemmer[3], ANNIE named-entity extraction[4] that give a good start.

Answer "yes" and you'll find that the system architecture become less intuitive. Language specific routines have to be implemented. What was straightforward monolingual solution now requires twisted multilingual algorithms. However, each new language that can be supported opens a new market and allows a new population to benefits from information extraction.

The following four sections present the basics requirement of information extraction for someone who decided to answer "yes" to the question above. It gives an overview of language detection, tokenization, sentence boundary detection and part-of-speech tagging.

2.2 Language identification

Apart from deploying algorithms that have to cope with multiple alphabet or language specificities, choice to be multilingual means the need for a language identifier that route your program toward the good dictionaries and sub-routines.

Here are two language identification strategies:

Stop Word Frequency Analysis:

> Given a text, count the number of stop words (e.g.: the, a, of, …) it has in each languages. The language for which there is the more Stop Word is likely to be the language of the source text. NRC's Extractor (Turney, 2000) uses this strategy.

N-Gram Frequency Analysis:

> The frequency of character N-grams is a good indicator of a text language. A text with a high frequency of the bi-gram "wh" is, indeed, more likely to be written in English than in French. Cavner and Trenkle (1994) present such an approach.

---

[2] http://www.cs.jhu.edu/~brill/
[3] http://www.tartarus.org/~martin/PorterStemmer/
[4] http://www.aktors.org/technologies/annie/

Even if it looks like a relatively easy task, there are open problems in this area. A problem is the identification of the language of really short texts. Consider for instance the one-word text: "Information". This text is at least valid in English, French and German.

Another problem is working with documents where segments of texts are in different languages. Also, documents with low level of language (slang, e-mails, Usenet) may be problematic.

The n-gram frequency analysis has an undeniable edge over stop word frequency analysis. It requires absolutely no dictionary and a minimal human expertise. It can be extend to almost any language only by feeding the algorithm with a sufficient amount of text. However, it may not scale as easily as the stop word method. Indeed, there is a clear difference between the stop words of two languages while the n-grams may be shared by many languages. Just take a look at the following n-grams that are excellent features but are shared by at least two languages:

- words that start with "qu" are common to French and Spanish;
- the sequence "sse" is common to French and German;
- the word endings "nd" and "ng" are common to English and German;
- the word ending "at" is common to English and Romanian;
- the word beginning "as" is common to English and Spanish;

It is conjectured that an optimal solution to language identification would be a mix of the n-gram and stop words frequency analysis.

2.3 Tokenization

The very first process when working with textual information is the tokenization. It consists in splitting a text in its constituent tokens like words, numeric expressions and punctuations.

This task is primordial enough to be implemented in standard libraries in many programming languages (ex.: Java, Python). However, many tokenization variants are imaginable and valid in various contexts. For instance, one may want to split an input stream on every white character and every punctuation. Another may prefer to keep strings with internal punctuations (usually http addresses, emails, fractional numbers and condensed dates).

Let's look at the most common issues of the tokenization process: hyphenation, unbreakable tokens, apostrophe, accentuated characters, ligature and canonical form.

2.3.1 On the Hyphen

Depending of the source of the input document, the hyphen can play an ambiguous role. Indeed, it can be use (1) in hyphenated words, (2) in pronominal inversions (ex.: "a-t-il"), but also (3) to split words at the end of sentence. In the third case, it is purely aesthetical and it would be useful to remove it. In the second case, it groups words with different part-of-speech (e.g.: a verb with a pronoun). Again it is preferable to remove it. Finally, (4) hyphen can be used as sentence apposition marker (replacement of the comma).

Depending of the application, hyphenated words can be considered as single tokens or multiple linked tokens. For instance, in full-text information retrieval, each part of hyphenated words has to be tokenized. It allows the retrieval of the word "ciel" in a document containing "arc-en-ciel".

2.3.2 On the Unbreakable Tokens

Unbreakable tokens are "pseudo-words" with the particularity to comprise punctuations or digits. For instance, trade marks ("7up"), organizations ("G-8") or programming language ("C++") may benefit from being count as single tokens instead of being tokenized in their constituents.

Again, there is an interest of keeping these words intact in information retrieval – that usually ignore punctuation – to differentiate, for instance, between a C++ and C# query.

2.3.3 On the Apostrophe

Apostrophe is a language specific issue to consider during tokenization. In English, the apostrophe can count as a possessive marker (e.g.: John's books, John' books) and as contracted words (e.g.: don't). In French, the apostrophe is always used to separate a determiner from a word that starts with a vowel (e.g.: l'avion) except in one single case: the word "aujourd'hui" that include an apostrophe.

The apostrophe also has a language independent behaviour. It appears in person surnames like "O'connor".

2.3.4 On the accentuated characters

English does not include accentuated characters except for some expressions borrowed from other languages (e.g.: *à la carte*, *bon appétit*, …). Obviously, accents are common in French but also in other languages like Spanish. The question that arises while tokenizing is whether or not the accents are to be kept. Simply said: Do we want the word "école" and "ecole" to be equivalent?

But why would we remove the accent? Because accentuated words are sometime voluntary unaccentuated (Editing an email with an English keyboard) or grammatically correct with or without the accent (accent on French capitalized letters is not mandatory).

In French, though, accents can be necessary to discriminate from two words with different meaning ("*né*" vs. "*ne*") or to identify verb tense. In Spanish, some accents are purely used for pronunciation purpose and can therefore be stripped without loss of meaning.

2.3.5 On the ligatures and special characters

Ligature is a problem specific to some electronic support. Ligature is a sequence of character that is encode as a single character. For instance, PDF documents uses ligature to render some characters. It is the case with the character sequence "*li*"

and "*fi*". Modern text editor will use the ligature in the word "*chœur*" where the "o" and the "e" are combined. Ligatures have to be decomposed in their constituent characters.

Another special character to consider is the German estsett "*β*". From an historical convention, it is equivalent to the sequence "*ss*".

Finally, special chars like copyright or trademark can be useful information and therefore benefit from being standardized. Unicode has a lot of reserved characters that represent variants of a same character (e.g.: the quote) or group of characters.

2.3.6 On the Canonical Form

Canonical means "reduced to the simplest and most significant form possible without loss of generality"[5].

For the purpose of dictionary lookup or for string matching operations, tokenization has to be done using some kind of canonical text version because, basically, lowercased characters can't match uppercased characters. Canonical form can be extended to cope with character case, accents, normalization of punctuation (ex.: ` vs. `) and ligature resolution.

2.4 Sentence Boundaries Detection

Sentence Boundary Detection (SBD) is often the next step, after the tokenization, in an information extraction system. SBD brings a great deal of challenges. Usually, sentences are separated by period, exclamation and interrogation marks. But these punctuations can also appear within a sentence without breaking it.

2.4.1 Common Difficulties with the period

The main reason why we cannot break sentence on each period is the abbreviations (e.g.: I'll leave at 5 p.m. tomorrow.) Another important difficulty comes from some special word (like trademarks) that sometimes uses punctuation (ex.: the Yahoo! trade mark). Let's also note the possible use of the period after ordinals in headers and sub-headers.

Given that we have a solid abbreviation identification system, the SBD task may look obvious to resolve. Unfortunately, there exist sentences that end with abbreviations (a phenomenon called haplology). In this case, the period of the abbreviation actually plays both the role of abbreviation mark and end of sentence mark.

One could propose that when an abbreviation is followed by a capitalized word, it breaks the sentences but what about "I met Mr. Jones."?

We could extend the rule to only break the sentence if the abbreviation is followed by a capitalized word of an unknown part-of-speech. It is rapidly becoming complex!

[5] http://dictionary.reference.com/search?q=canonical

2.4.2 Quotation mark and Parenthesis

A lot of problems arise from the use of quotation marks and parentheses. The period that ends a citation may or may not end the sentence as well. For example:

- *The president said: "We will create jobs." He also talked about taxes.*

Here, the period placed after the word "job" plays the role of the end of the citation and the end of the sentence.

An important challenge, in addition, is the case of sentences within sentences. Just take a moment to identify the sentences in the example below:

- *At daytime, the zebra is easy to see because of its black lines (Author Note: I have no evidence of this claim. But I found it was a funny example.) while at night, we can easily distinguish its white ones.*

Finally, the "period-quote" sequence is not obvious to resolve. There are two possible cases of resolution:

1) A quote that ends a sentence:
   He said: "Here are my remarks**."**

2) A quote that starts a sentence:
   That's what he said**. "**Here are my remarks", he also adds.

To resolve such cases it seems to be necessary to know where each quotes starts and ends. It can be really tricky when there are embedded quotes.

2.4.3 Titles and Fragments

Sentences that do not end by punctuation (like titles) and fragments found in bulleted lists, tables, etc. are another common problem. In this case, it is necessary to detect that a new sentence is starting on a new line. However, the "new line character" is ambiguous because it can be use anywhere for formatting.

Here's an example of two lines, separated by a new line character, that belong to the same sentence:

> *Hello, this is a simple mail to remind you that we should meet*
> *Tuesday at Ottawa University.*

Here, the lines are two independent sentences:

> *Washington Post, May 3rd 2004*
> *Tuesday, the White House officials declared that…*

2.4.4 Worst Case Scenario

Some worst case scenario makes us realize that SBD remains an open problem without the contribution of complex semantic routines. Just consider the two sentences below:

- I must be at the garage at 5 p.m. Sunday to fix my car.

- I must be at the garage at 5 p.m. Sunday is the day I choose to fix my car.

In the first sentence, the period after p.m. does not count as a sentence boundary. In the second case, it does.

2.4.5 More language specific issues

In the last section, it has been noted that title recognition is important to split sentences that does not end with a period. In English, the capitalization is a reasonable evidence of a title. In French, however, it is not common to capitalize each word of titles.

In Spanish, there is an additional cue. Indeed, the interrogative and exclamative sentences start with an inverted punctuation.

In German, every noun is capitalized and therefore cannot be use to guess sentence boundaries after an abbreviation or on a new line. Also, German numerals in date expressions are followed by a period (ex.: 1. Januar). German, again, is a verb-final language and this particularity could be really helpful in SBD.

There's finally some regional difference in numbers standards, In English, decimal is denoted by the period (5.2) while in French, the comma is often use (5,2). The former case is an additional source of period ambiguity.

2.5 Part-of-Speech Tagging

A task that received a great deal of interest in the scientific communities is the part-of-speech (POS) tagging. POS consist in the attribution of a part-of-speech (noun, verb, adjective, etc.) to each token. It can be seen as basic information to gather since a lot of background work is available and relatively simple techniques can achieve high precision.

Many tasks will eventually benefits from the identification of noun, adjective and verbs at different levels of granularity. For instance, it can be useful to know verb tense or adjective gender.

We'll not detail further the task of POS since it is a research field by itself and go beyond the scope of this essay. In the Balie system, presented in the next section, the POS tagging is performed by an external product called qTag (Mason…reference).

3 Balie

3.1 Baseline Information Extraction

Balie is a system for multilingual information extraction from textual sources. Basically, it provides the following services:

- Language identification;
- Tokenization;
- Sentence boundary detection;
- Part-of-speech tagging;

The following section briefly motivates Balie and compares it with other information extraction systems. Then, section 3.3 presents implementation details of Balie. Section 3.4 presents its class diagram and, finally, section 3.5 gives some sample usage of Balie with Java code.

Balie is in Java and is open source software issued under the GNU General Public License. It is hosted at SourceForge[6].

3.2 Motivation

Balie's goal is to offer baseline architecture for a wide range of information extraction tasks. The difference between Balie and other tools like Gate[7], Oak[8] or MinorThird[9] is in its machine learning foundation and multilingual architecture. Indeed, Balie implements no rule system and it encodes a minimum of human knowledge. It is instead based on the Weka toolkit (Witten and Frank, 2000). Other systems, for instance, typically resolve the sentence boundary problem by hand-crafted rule system. In Balie, the solution is learned and the human expertise is limited to annotate a corpus with sentence boundary information.

Here's a qualitative overview of the tree systems above. No evaluation was made on the functionality itself but it could be seen as a future work item.

3.2.1 Gate

Gate is a large NLP system. The core components of Gate are designed to facilitate the corpus linguistic by putting together corpus manipulation/annotation tools, dictionary manipulation, NLP algorithms and visualization tools. Clearly, it won't serve the same purpose as Balie.

A platform called "Annie" was developed using Gate. This piece of software offers the same functionality as Balie: tokenizer, sentence splitter and part-of-speech tagger. However, it seems to be mainly a monolingual English system with human-engineered rule-based systems. Advanced features like co-reference resolution and named-entity recognition are offered by Annie.

---

[6] http://balie.sourceforge.net/
[7] http://gate.ac.uk/
[8] http://nlp.cs.nyu.edu/oak/
[9] http://minorthird.sourceforge.net/

3.2.2 Oak

Oak system is well described in its project page: "OAK system is a total English analyzer, which consists of a sentence spliter, a tokenizer, a POStagger, a stemmer, a chunker, a Named Entity (NE) tagger, a dependency analyzer, a parser, a function tagger and a regularizer. It basically uses explicit rules, rather than probabilistic scores, so that human can modify and hopefully improve the accuracy. The rules are mostly extracted based on transformation or decision list learning method, and the rules are look like regular expressions."

The differences between Oak and Balie are well expressed: monolinguality vs. multilinguality and rule-based vs. machine learning.

3.2.3 MinorThird

MinorThird combines tools for annotating and visualizing text. It is not explicitly concentrating on multilingualism and does not offer machine learning solution to sentence boundary detection, for instance. It, instead, concentrates on learning methods for learning to extract and label spans from a document, or learning to classify spans.

Overall, the three systems presented above differ from Balie on the multilingual aspect and the learned solution to basic IE requirements. Note that they all offer more advanced features than Balie and, in the case of Gate and MinorThird, also offer a visual tool to manipulate and annotate corpus and dictionaries.

However, due to the software engineering choices, we believe that these systems are easier to use through their GUI as experiment tools while Balie is easier to be programmatically use and included in advanced IE systems.

3.3 Balie Implementation details

The atomic unit in Balie is the Token – a word or a punctuation taken in isolation. When a text is processed, it is turned in a list of Tokens (*TokenList*) – a list of words and punctuations. The output *TokenList* can either be manipulated in memory or through an XML representation. The machine learning routines of Balie are delegated to Weka (Witten and Frank, 2000).

The following subsections cover the three mains structures of Balie: the *TokenList*, the XML representation and Weka automation.

3.3.1 Balie Token and TokenList

The *TokenList* is an ordered bag of tokens. It represents each words and punctuation of a source text with all the information required for advanced information extraction. Each token is defined as follow:

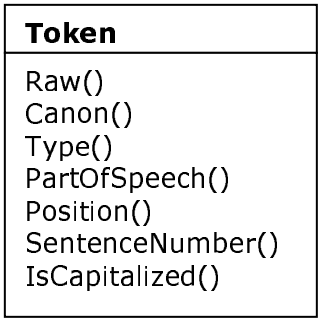

Figure 1: The token, atomic unit of Balie.

The *Raw* version of the token is how it appears in the source text, with capitalization, internal punctuations, etc. The *Canon* version of the token is, on the other hand, lowercased, without internal punctuations and with punctuation normalized (see section 1.2.7). The *Type* is either "word" or "punctuation". *Position* (in number of tokens), *SentenceNumber* and *IsCapitalized* properties are self-descripting. *PartOfSpeech* is quite an important property and it is designed to be use with the bit mask technique. It is represented on a 32-bits integer. Each position is reserved for a given part-of-speech. For instance:

Bit position	Part-of-speech
1 (rightmost)	Determiner
2	Noun
3	Adjective
4	Verb

Table 1: Meaning of some bits in the Part-of-Speech integer value.

In Java, this is simply defined using the shift operation:

```
// Part of Speech
public static final int POS_DETERMINER                    = 1 << 0;
public static final int POS_NOUN                          = 1 << 1;
public static final int POS_ADJECTIVE                     = 1 << 2;
public static final int POS_VERB                          = 1 << 3;
…
```

Figure 2: Declaration of Part-of-Speech values in Java.

The interest of such a representation is the possibility to test many possibilities at once using a bit mask. For instance, to test whether a given token is part-of-speech noun or verb, the bitwise AND test can be applied:

Noun Token	0000…0010
Bit mask	0000…1010
Bitwise AND	0000…0010 (true)

Figure 3: Testing if a token is a Noun or a Verb.

Moreover, this representation is useful to represent ambiguity, since a token can have multiple part-of-speeches at once by raising many bits.

The *TokenList* has multiple entry points. It is basically a list, that can be sequentially accessed, but it also implements a hashtable, for random access, and a TermFrequency map that keeps track of the frequency of each unique canonical token.

Entry point	Use	In Java
List of tokens	Sequential access to each tokens	`ArrayList m_TokenList`
Index of tokens	Random access to any token	`Hashtable m_HashAccess`
TermFrequency	Frequency of each unique canonical token	`Hashtable m_TermFrequency`

Table 2: Various entry points for the tokenlist.

3.3.2 XML representation

The *TokenList* can either be manipulated in memory or through an XML representation. The latter option is a useful interchange format or a visualization tool. Below is the XML representation of the *TokenList* for the sentence "*Information Extraction (IE) is the name given to any process which selectively structures and combines data which is found in one or more texts*."

```
<?xml version="1.0"?>
<balie><tokenList><s>
<token type="2" partofspeech="2" canon="information">Information</token>
<token type="2" partofspeech ="-2147483648"
canon="extraction">Extraction</token>
<token type="1" partofspeech ="64" canon="(">(</token>
<token type="2" partofspeech ="16" canon="ie">IE</token>
<token type="1" partofspeech ="128" canon=")">)</token>
<token type="2" partofspeech ="8" canon="is">is</token>
<token type="2" partofspeech ="1" canon="the">the</token>
<token type="2" partofspeech ="2" canon="name">name</token>
<token type="2" partofspeech ="8" canon="given">given</token>
<token type="2" partofspeech ="64" canon="to">to</token>
<token type="2" partofspeech ="1" canon="any">any</token>
<token type="2" partofspeech ="2" canon="process">process</token>
<token type="2" partofspeech ="1" canon="which">which</token>
<token type="2" partofspeech ="16" canon="selectively">selectively</token>
<token type="2" partofspeech ="2" canon="structures">structures</token>
<token type="2" partofspeech ="32" canon="and">and</token>
<token type="2" partofspeech ="8" canon="combines">combines</token>
<token type="2" partofspeech ="2" canon="data">data</token>
<token type="2" partofspeech ="1" canon="which">which</token>
<token type="2" partofspeech ="8" canon="is">is</token>
<token type="2" partofspeech ="8" canon="found">found</token>
<token type="2" partofspeech ="64" canon="in">in</token>
<token type="2" partofspeech ="536870912" canon="one">one</token>
<token type="2" partofspeech ="32" canon="or">or</token>
<token type="2" partofspeech ="1" canon="more">more</token>
<token type="2" partofspeech ="2" canon="texts">texts</token>
<token type="1" partofspeech ="1" canon=".">.</token>
</s></balie></tokenList>
```

Figure 4: XML representation of the TokenList.

3.3.3 Weka automation

The machine learning capabilities of Balie are based on Weka (Witten and Frank, 2000). This is done by manipulating feature vectors directly in Weka and by saving learned models on disk. Balie uses two main classes to interact with Weka: The *WekaLearner* and the *WekaPersistance*. The following figure shows a class diagram that summarizes the interaction:

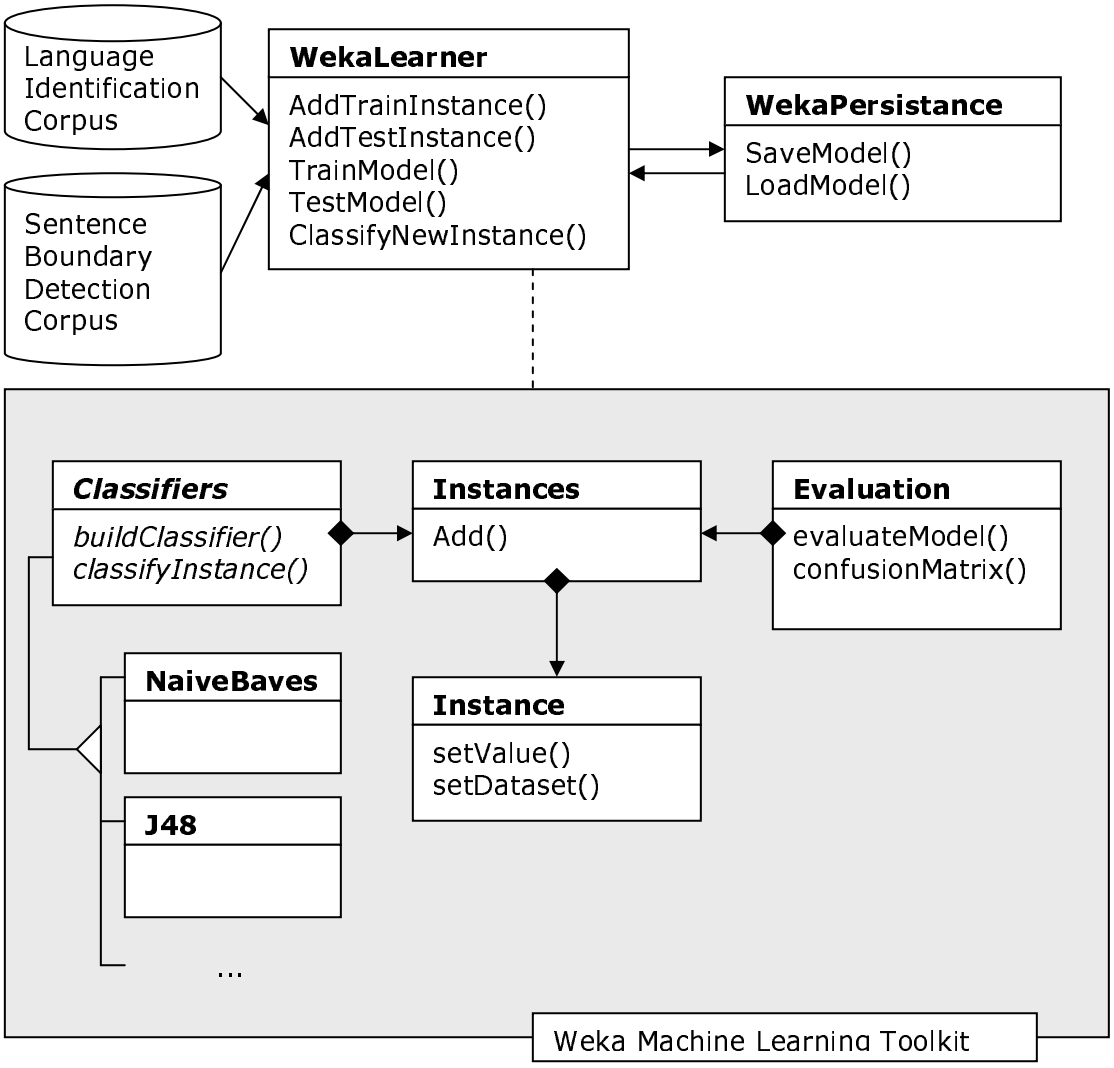

Figure 5: Simplified class diagram for interaction between Balie and Weka.

3.4 Class Diagram

The previous sections gave a great deal of attention to the *TokenList* structure. This is, clearly, the central structure of Balie. The *TokenList* is created by the *Tokenizer*. The following figure is a simplified class diagram of Balie, centered on these elements:

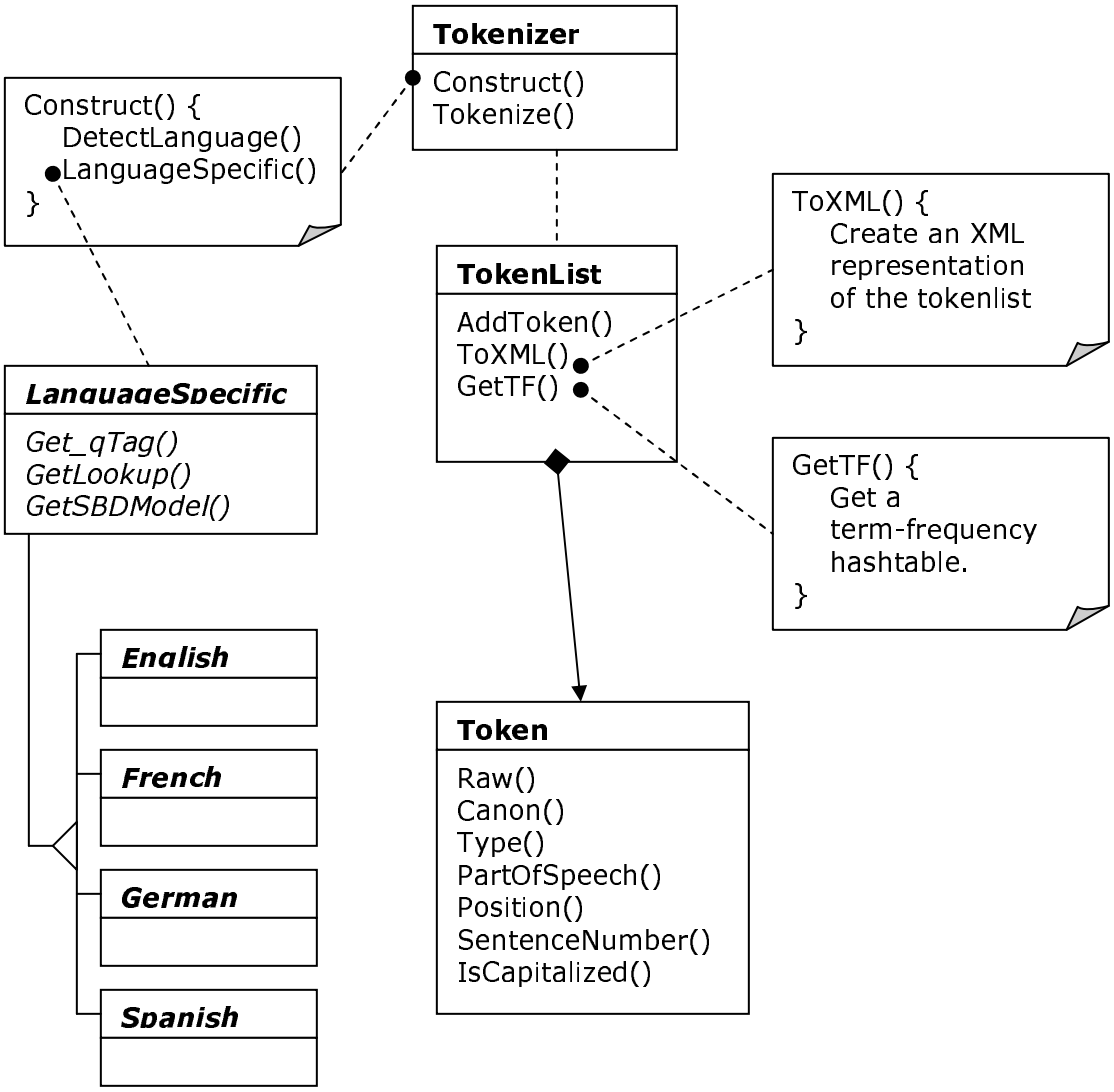

Figure 6: Simplified class diagram for Balie.

The *Tokenlist* can aggregate one or many tokens. The *Tokenlist* is created by the *Tokenizer*. This module will guess the language before starting to process the text. It also loads in memory all the language specific lookup table (dictionaries) and routines. It finally uses the SBD model to split sentences.

3.5 Sample usages

3.5.1 Using Balie

This sample java code will read a French text, number the sentences and put verbs between squared brackets.

```
import ca.uottawa.balie.TokenList;
import ca.uottawa.balie.Tokenizer;
import ca.uottawa.balie.LanguageIdentification;
import ca.uottawa.balie.Token;
import ca.uottawa.balie.TokenConsts;

/**
 * @author nadeaud
 */
public class sample {

    public static void main(String[] args) {

        String strText = "Les médiateurs cherchent une issue à la crise\n\nAnia
Tsoukanova\n\nAgence France-Presse\n\nKiev\n\nLes manifestants d'opposition ont
bloqué vendredi en Ukraine les entrées au siège du gouvernement, alors que
plusieurs médiateurs étrangers, dont le représentant de l'UE Javier Solana et
le président polonais Aleksander Kwasniewski, arrivent à Kiev pour tenter de
désamorcer la crise. "+
                                                                "Le premier ministre
Viktor Ianoukovitch, vainqueur contesté de la présidentielle de dimanche, a été
empêché de se rendre à son bureau, a annoncé sa porte-parole. Faisant monter la
pression, plusieurs milliers de manifestants ont bloqué le siège du
gouvernement et renforcé le blocage de la présidence entamé la veille, formant
un cordon étanche autour du quartier tout entier. Des autobus et des minibus
ont été placés devant l'entrée principale du siège du gouvernement, des
milliers de manifestants encerclant le bâtiment pour empêcher les
fonctionnaires d'y entrer. "+
                                                                "Une dizaine de
policiers se trouvaient à l'intérieur de l'enceinte, mais on n'observait pas de
tension particulière. Un groupe de manifestants, agitant des drapeaux oranges
qui font voler les flocons de neige, entonnent des chants populaires, alors que
les conducteurs de voitures passant à proximité klaxonnaient bruyamment pour
manifester leur soutien. Parallèlement, les rumeurs sur l'arrivée à Kiev de
nombreux avions russes pouvant transporter des hommes armés sont devenues
persistantes.";


        // Load language identification module
        LanguageIdentification li = new LanguageIdentification();

        // Create a tokenizer for the appropriate language
        Tokenizer t = new Tokenizer(li.DetectLanguage(strText), true, true);

        // Tokenize the text
        t.Tokenize(strText);

        // Obtain the token list
        TokenList tokenList = t.GetTokenList();

        // Print sentences, with verb in squared brackets
        int curSentence = -1;
        for (int i = 0; i != tokenList.Size(); ++i) {
            Token curTok = tokenList.Get(i);
            if (curTok.SentenceNumber() != curSentence) {
                curSentence = curTok.SentenceNumber();
                System.out.println("");
                System.out.print(String.valueOf(curSentence)+"- ");
            }
            // open squared bracket if verb
            if (TokenConsts.Is(curTok.Type(), TokenConsts.TYPE_WORD)) {
                if (TokenConsts.Is(curTok.PartOfSpeech(),
TokenConsts.POS_VERB)) {
```

```
                    System.out.print("[");
                }
            }
            // print token except newline
            if (!TokenConsts.Is(curTok.Type(), TokenConsts.TYPE_PUNCTUATION)
||!TokenConsts.Is(curTok.PartOfSpeech(), TokenConsts.PUNCT_NEWLINE)) {
                System.out.print(curTok.Raw());
            }
            // close squared bracket if verb
            if (TokenConsts.Is(curTok.Type(), TokenConsts.TYPE_WORD)) {
                if (TokenConsts.Is(curTok.PartOfSpeech(),
TokenConsts.POS_VERB)) {
                    System.out.print("]");
                }
            }
            System.out.print(" ");
        }

    }
}
```

Figure 7: Java code for a simple Balie usage.

3.5.2 Learning with Weka

This sample java code will learn a model from instances made of four double values. It then test the model, classify a new instance, save the model on disk, load it back and classify another instance. The program demonstrates the use of the WekaLearner and WekaPersistance classes.

```
WekaAttribute[] wekaAttr = new WekaAttribute[]{
       new WekaAttribute("Double1"),
       new WekaAttribute("Double2"),
       new WekaAttribute("Double3"),
       new WekaAttribute("Double4")
};
String[] strClass = new String[]{
       "Positive",
       "Negative"
};

WekaLearner wl = new WekaLearner(wekaAttr, strClass);

wl.AddTrainInstance(new     Double[]{new     Double(0.0),new     Double(1.2),new
Double(2.2),new Double(4.4)}, "Positive");
wl.AddTrainInstance(new     Double[]{new     Double(1.0),new     Double(1.0),new
Double(7.7),new Double(1.9)}, "Negative");
wl.AddTrainInstance(new     Double[]{new     Double(0.1),new     Double(0.0),new
Double(0.0),new Double(0.0)}, "Positive");
wl.AddTrainInstance(new     Double[]{new     Double(1.1),new     Double(1.1),new
Double(1.1),new Double(1.1)}, "Positive");
wl.AddTrainInstance(new     Double[]{new     Double(3.4),new     Double(9.9),new
Double(0.2),new Double(1.0)}, "Positive");
wl.AddTrainInstance(new     Double[]{new     Double(1.5),new     Double(0.1),new
Double(7.0),new Double(0.0)}, "Negative");

wl.CreateModel(new NaiveBayes());
```

```
wl.AddTestInstance(new     Double[]{new     Double(1.5),new     Double(0.1),new
Double(7.0),new Double(0.0)}, "Negative");
wl.AddTestInstance(new     Double[]{new     Double(1.1),new     Double(1.1),new
Double(1.1),new Double(1.1)}, "Positive");

System.out.println(wl.TestModel());

double  x  =  wl.Classify(new  Double[]{new  Double(3.4),new  Double(1.0),new
Double(9.9),new Double(0.0)});
System.out.println(String.valueOf(x));

WekaPersistance.Save(wl, "c:\\test.weka");
wl = WekaPersistance.Load("c:\\test.weka");

x    =    wl.Classify(new    Double[]{new    Double(3.4),new    Double(1.0),new
Double(9.9),new Double(0.0)});
System.out.println(String.valueOf(x));
```

Figure 8: Java code for a simple interaction with Weka.

4 Conclusion

This report presents a system for multilingual textual information extraction (IE) called Balie. It also gives an overview of four basic algorithms required for IE on texts: language identification, tokenization, sentence boundary detection and part-of-speech tagging.

Some issues of those four processes are exemplified and discussed. The implementation choices, in Balie, will certainly be more or less compatible with each IE tasks. For example, it is better to tokenize all parts of compound words for information retrieval while it is handy to keep them as is for dictionary lookup. Also, it may be better to strip accents in canonical form of words in some scenarios even if it is risky to loose the right sense in other scenarios.

It is shown, through examples, that there are still many open problems, even in the basic areas of information extraction we cover. For instance, language detection will not be accurate on short texts or text with multiple languages (section 2.2). Moreover, sentence boundary detection will certainly be fooled by some worst case scenarios (section 2.4.4).

The Balie system is presented as an attempt to solve the basic requirements of information extraction. The architecture of Balie is centered on the idea of *TokenList* (section 3.4). This is a concept shared by other systems like Gate, Oak or MinorThird. However, Balie is entirely based on machine learning and concentrates on multilingualism. Other systems are typically designed around human-engineered rule-based systems (section 3.2) and are offered as work environments. Due to the software engineering choices, we believe that these systems are easier to use through their GUI as experiment tools while Balie is easier to be programmatically use and included in advanced IE systems.

It is also believe that a machine learning approach to information extraction will reduce the cost of system maintenance and facilitate the integration of new languages. Most of all, the trainability of Balie appears as a way to reuse the same basics component to many advanced IE tasks.

5 References

Cavner, W. B. and Trenkle, J. M. (1994) N-gram based text categorization. *In Proc. of the Third Annual Symposium on Document Analysis and Information Retrieval*, pages 161-169.

Tufis, D. and Mason, O. (1998). Tagging Romanian Texts: a Case Study for QTAG, a Language Independent Probabilistic Tagger, Proceedings of the First International Conference on Language Resources and Evaluation (LREC), Spain, p.589-596

Turney, P.D. (2000), Learning algorithms for keyphrase extraction, *Information Retrieva*l, 2 (4), 303-336.

Witten I, H, and Frank, E. (2000) *Data Mining: Practical machine learning tools with Java implementations,* Morgan Kaufmann, San Francisco.