Final Project Report

Information Extraction 2007

1/13/2007

Sebastian Martin, Benjamin Hunziger, Dorian Schneider, Mykola Panasenko

Table of Contents

1) Overview

This is the final report for the project of the Information Extraction course held 2006/2007 at the Jiao Tong University, Shanghai, by Professor Tianfang Yao. This report will explain the task, our work done and our conclusions in the following chapters.

a. System Function

The task which should be completed was to write an automatic Information Extraction System. A website which could be chose by each team individually should be parsed and its contest should be examined to recognize the following 4 Named Entities:

- Date & Time
- Location Names
- Organization Names (Event Names)
- Person Names

This information should then be displayed in a user readable format like:

[ON Juventus], [ON Lyon] advance in Champions League [PN Alessandro del Piero] of [LN Juventus] celebrates his winning goal against [ON Bayern Munich].

The website chosen by our team is http://www.tennis.com (1). Tennis.com is one of the most famous news sites about tennis. Our task was to perform the information extraction for the headline news of this website.

b. Running Environment

As we chose a platform independent programming language with Java, the running environment is more or less platform independent as well. All platforms supporting Java can run our program. The necessary Java version is Java 1.5 (2), also running environments we tested are platforms with the newest version of Java 1.6.

c. Development Environment

The development environment has been different machines running Microsoft Windows XP and Windows 2000. The development software used was Eclipse 3.1, Eclipse 3.2 (3) and

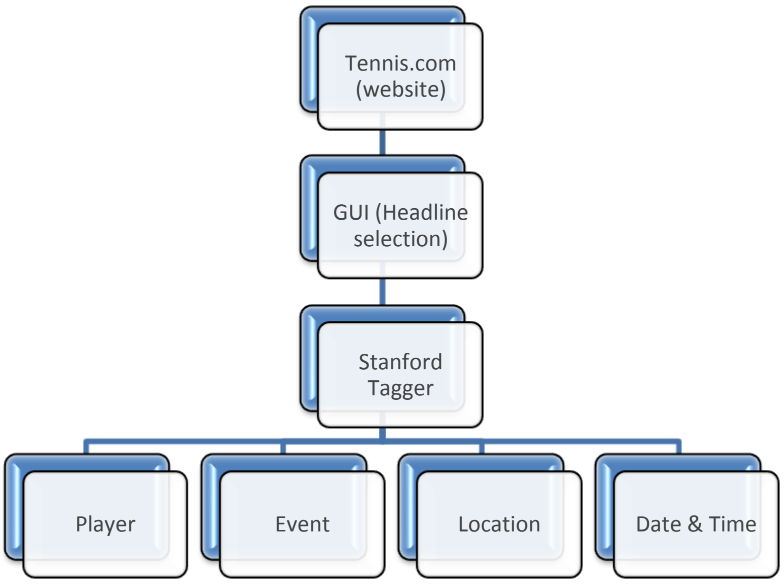
IntelliJ (4). The different environments are the result of the different tasks and environments preferred by each team member.

2) Task Arrangement

As described before, the required tasks have been according to the different Named Entities: Date & Time, Location Names, Organization Names (Event Names) and Person Names. The tasks where split so each team member processed one Named Entity. Dorian Schneider worked on Date & Time recognition and the GUI of the program as well, Mykola Panasenko on Location Names, Benjamin Hunziger on Organization Names (which in our case was considered to be the Event Name of a Tournament), and Sebastian Martin extracted Player Names from the News. The other tasks like the Presentation and Documentation were equally divided in between all the group members.

3) System Architecture

As the Named Entities in the case of Tennis.com where limited (like a global static list of Tournaments for each year, a list of Players who participate in the famous Tournaments and therefore are mentioned in the news), we decided to create a System with Dictionaries and Rules rather than implementing a stochastic system like Markov Model with a lot of learning procedures. Our system architecture consists of:



The website Tennis.com is read automatically by our Graphical User Interface. The user then can select the headlines to be parsed. These selected headlines (and the full body text of the news) will be passed to our integration of the Stanford Tagger (5). The tagged text is then forwarded to each of the four modules for the different Named Entities like Location Extraction, Player name Extraction etc. These modules parse and examine the text and return the results to the GUI which displays the final result in its own window. Each model itself has its own set of rules, dictionary files and methods to process the text that will be returned as result. Through this modular approach, each module can as well separately be activated or deactivated according to users needs or could be exchanged with different modules or versions without affecting the main program and GUI.

4) Algorithms

This section describes the algorithms used by the different Modules.

a. Date & Time

The Date and Time extraction module

Introduction

This class is part of the Xtractor information gathering program and constipated to locate and extract date and time information provided by the tennis.com news system.

The module consists of two java classes : TDMain. Java and Library.java. TDMain, the core of the engine, contains the necessary methods to perform the search and extraction tasks. On the other hand, the Library class operates as an dictionary and rule set container at once, providing the needed information and rule patterns used by the extraction algorithm to differentiate between true date and time formations from those who are not.

To meet those requirements, the TDMain class performs four serial steps to filter the desired information from the given, pre-tagged text string. Those steps are discussed in detail in the following pages by briefly outlining the class architecture (further information should be taken from the source code comments)and describing the main methods and member variables. Afterwards, we will describe the routines and mechanism used to find isolate the content of interest.

Class Organization

Class TDMain. Java

```
public Vector<String[]> extractDateAndTime(String argz){}
```

The main method. Receiving the to analyze string, containing the main loop and calling all other filter methods.

```
private  Vector <Object[]> filterByRules(){}

      First filter method. Applying the rule sets to the word
      constellation.


private boolean checkFiltered( Vector<Object[]> filteredElements){


      The second filter method. This time the content is not filtered
      by rules, but handed to the dictionary filter methods



private boolean isNounContentOfInterest(String text) {}

      First dictionary filter method. Checks if the word (non time
      word i.e. non /CD tagged) is part of the dictionary.



private boolean isNumberContentOfInterest(String text) {}

      Second dictionary filter method. Checking if the found word
      (number word, i.e. tagged by /CD) is formatted like an usual
      date or time
```

Class Library. Java

```
public  String[] days

      A string array containing a list of the seven days

public  String[] months
```

A string array containing a list of the twelve months

```
public  String[] digits
```

A string array containing a list of written numbers from one to ten

```
public  String[][] rules
```

```
A string array containing a set of grammatical rules, created
to analyze and judge specific sentence constellations
```

```
public  String[] specificTxt
```

`A string array containing` words that are often used in a date or time context

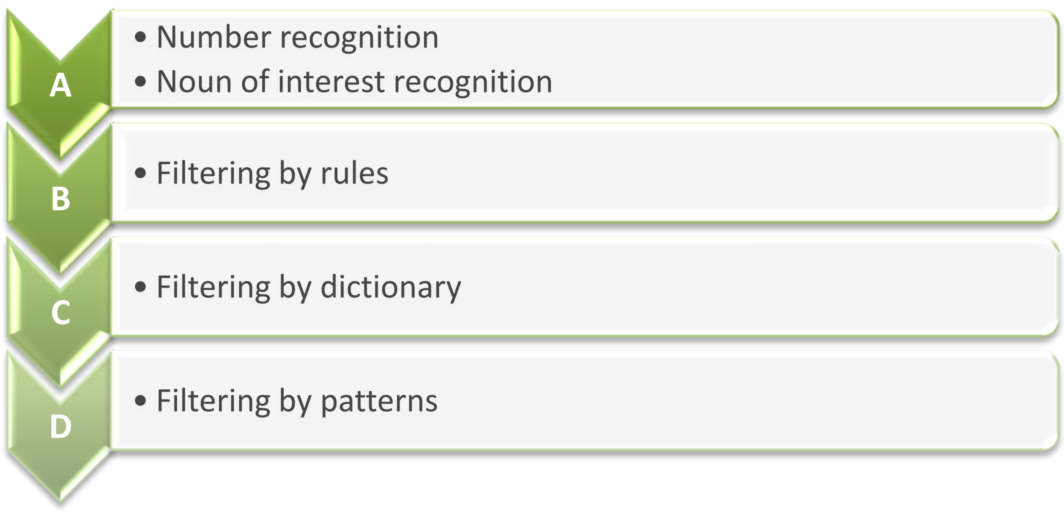
```
public DateFormat[] dateFormer
```

```
A date format array, containing typical date formatio patterns
```

```
public DateFormat[] timeFormer
```

```
A date format array containing typical time formation patterns
```

The Four Steps To The Throne



Step One

Find the Target

To begin with, the text is parsed using a loop who runs as long as the text is long and until all the words contained in the string have been processed. Five words are stored in a text buffer who is realized by a five element string array. Each loop the words in the buffer are shifted one position to the left, the fifth position is taken by the new word. The analyzed word is located at position three in the buffer, namely the middle of the array.

The first if-statement tests if the word of interest (in the following named WOI) is tagged with /CD (Stanford Tagger acronym for a number) or if it I part of the 'Date and Time' dictionary. If this condition is satisfied, the second if-statement takes place, if not, the buffer is updated with a new word and a new loops begins.

Furthermore, a second if-statement verifies if the identified number or number-word is element of a word constellation that leads to a high probability for WOI to be a date or time indicator. However, if the WOI fits to one of the pre-defined date patterns, all rules are ignored and the word is stored to the output vector

Step Two

Preselect the candidates

The rule testing algorithm first makes a snapshot of the current grammatical word structure by analyzing the word tags in the buffer and storing them separately in a 'constellation' array. This array serves as template for the rule filtering process. By comparing the c-array to every single rule in the database, the program eliminates the candidates that have word structures with a low 'date and time content' probability. The rules are defined in an array, where each rule is composed by five elements, each element representing a text buffer position and containing a tag acronym. Elements with wildcards indicate that the word position is irrelevant for the rule to take place.

Example:

```
Rule : { {"**", "DT", "CD", "**",    "**" }
```

Fitting constellation:

… began at 16:30 and ended….

Non-fitting constellation:

… defeated with 7:6 showing a….

Step Three

Validate the filtrate

Rules can never satisfy our demanding expectations in a perfect way. Rules that are not strict enough let pass too many candidates trough the barrier- if the conditions are too strict, some real date or time indicators won't be treated. For overcome this limitation , we integrated a second filter mechanism who is based on a dictionary recognition method, and further validates the candidates preselected by the rule-based filter. In fact, at this position the sentence constellation is not important any more but only the WOI itself. If the WOI is not a number the program checks if the WOI is an element of the library, in terms of eliminating the remaining non-real date and time filtrates.

Step Four

Pattern Checking

The final step consists of testing if the number WOI fits to one of the predefined date or time patterns. In general, a date is formatted in a strict and standardized format. European people define dates in a 'dd.mm.yyyy' (d = day, m = month, y = year) formation, where US people does it in a 'mm.dd.yyyy' formation. Those pattern are very typical and dedicated to dates, so if the program finds and verify such an number formation, the highest priority is attributed to the WOI and it is automatically stored in the output vector without any further rule application.

Result Storage

The remaining filtrates are considered as real date or time indicators and store to the output vectors together with their position relative to the text. The final vector is given to the calling parent function and the results are displays using the provided GUI methods. My tests revealed, that more than 89% of the D&T indicators are found and extracted correctly. The module is easily extendible by adding new rules or adding new data to the dictionary, in other words, the performance can even be improved by doing some statistical rule research and / or addition new dictionary words.

b. Location Names

i. Class

The „Location Entity Recognition“ module mainly consists of 1 class called „extractLocation“, which itself can be found in the extractLocation.java file. This class consists of several methods which contribute to the proper Entity extraction. The main method is called „doExtract(String input)“ which takes a tokenized String as input and returns a Vector containing String Arrays. These arrays consist of the extracted word and its position in the text.

An the following example will illustrate it:

doExtract("Russia/NNP and/CC Germany/NNP advances/NNS to/TO Hopman/NNP Cup/NNP final/JJ.");

Output (Location,Position): *[("Russia","1"), ("Germany","3")]*

ii. Database

The database containing cities where both ATP and WTP Tournaments are hold and can be found in the main directory under the name of „cities“.

On the other side the file holding countries can be found under the name of „states“.

The Database format is simple, and each line contains an entry of the database.

Here the first 5 lines from both files:

Cities	Countries
Hertogenbosch	**Afghanistan**
Acapulco	**Albania**
Adelaide	**Algeria**
Amelia Island	**Andorra**
Antwerp	**Angola**

„Cities“ totally holds 189 elements.

„States“ totally holds 92 elements.

iii. Algorithm

First there steps which are performed when calling the doExtract() Method will be briefly described below.

- Remove redundant information like line breaks and whitespaces
- Read the database into memory and adjust it to a suitable format (Vector representation)
- Convert the tagged input string into a suitable Vector format for further processing

Next the algorithm iterates over every proper noun of the input text and performs the following steps:

- Every proper noun in the text is matched against every city and country from the database
- Words containing elements from the database not starting at its first position are ignored (e.g. S***albania*** which is a name contains the country name "Albania" , but is of course not a country)

In general, the rule for matching a Location is:

WORD BEGINS WITH LOCATION + "." OR +"," + "'" OR +"s"

- Words like „**Australia**'s","**Australias**", "**Germany,**" are detected
- Words like „**Australi**an" are discarded, because they do not match the rule and furthermore do not represent a location

iv. Optimizations

- Database only hold locations related to Tennis and Tournaments (could be enlarged to a arbitrarily huge one, even holding all villages of Africa)
- Only proper nouns are taken into consideration for further processing

v. Results

Due to the format of the news on tennis.com, which mostly reports on happenings related to tennis tournaments, nearly every location is extracted correctly by this module. Nevertheless, due to its database based extraction methods, there will be obviously entities which cannot be extracted. One just needs to report about an Event which is not an official ATP or WTA Tournament.

Despite of this, results with a success rate over 95% are achieved.

c. Organization Names (Event Parsing Moule)

The event parsing module is one of the currently four implemented parsing modules. This section describes its functionality, its methods, the data structures and the sequence of work. It works on a "word-for-word" based parsing level and makes use of two separate dictionary files. One dictionary includes a list of current tennis events; the other dictionary holds a list of what we call "special events". These are event extensions, such as "final" or "quarterfinals" that are often used after a general event expression in a news article. Its use will be explained later on in this chapter. These dictionaries do not have a specific length and can be extended or shortened in an arbitrary way.

i. Module Functions

The parsing module, which is included in the java class EventParser.java consists of eight methods that provide the needed functionality of parsing the given texts and return the wanted output. These methods are being described separately and in detail in this section. All of the described methods are called in a defined sequence by a method "parse(String file)" that receives the defined string, holding the tagged news event, and returns a vector holding the wanted information about the found events. The specific order of calling the other methods is described in the next section.

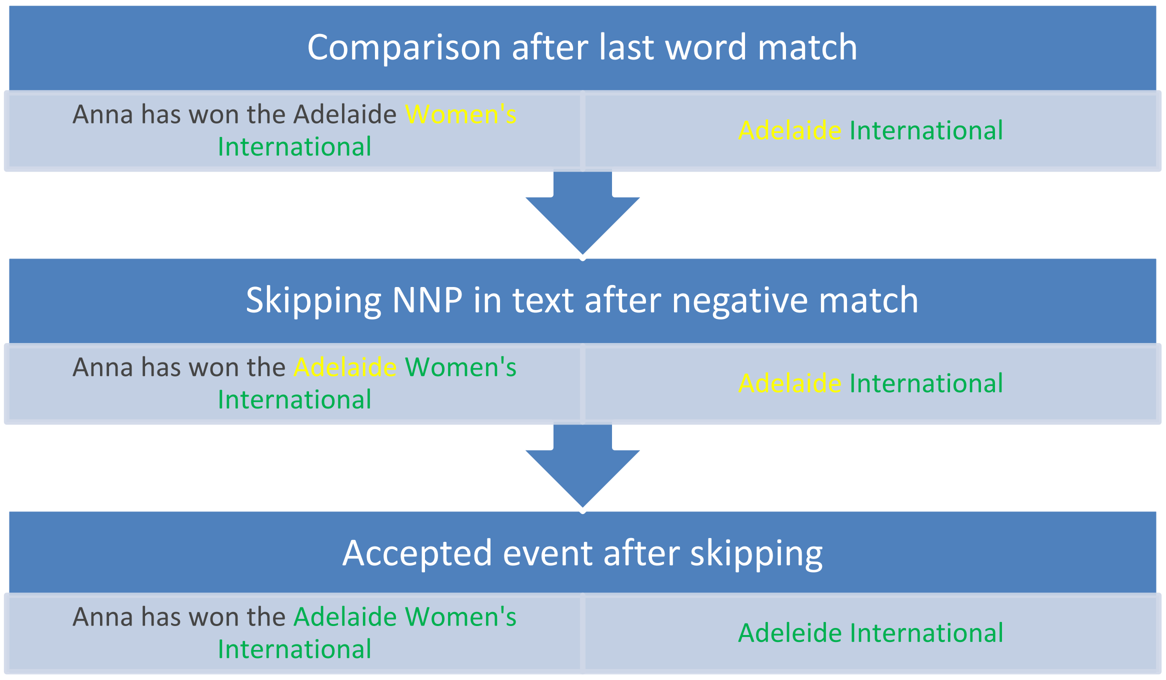
For testing purposes, there is a deprecated method "readInputFile(String filename)" that reads a text file holding a news article that has already been tagged by the Stanford tagger. This method is not used in the final version of the entire program, but was used to test the event module by itself. The method reads the text from the file system and converts it into a single string that is returned by the method. This string is then of the same format as the string, that is passed to the module later when the module is being attached to the final program.

The method "tokenizeInputFile(String file)" is the first method to prepare the input string that is passed to the event module for the actual parsing act. The method divides the text into triples of word, Part-of-Speech tag and word-position in the text. These three items are written into an array that now represents all information about the current word. This array is then appended to a vector that will finally hold all words of the read text. This vector is returned at the end of the method.

Before the prepared input data can be processed by the parser, the module needs to read and prepare its dictionaries for comparison. Therefore the method "readDatabaseFile(String filename)" is called, which reads the file containing the list of current tennis events. The file contains one event per line. The method therefore adds each line to a string, divided by a "/" for later preparation. The string containing all events is returned by the method. To further process this string the method "tokenizeDatabaseFile(String file)" is called. It works in a similar matter as the method that tokenizes the input file. The events are being separated by the "/" and then split into their single words. The single words that belong to one event are added to a vector. This vector now represents one event of the database. All event-vectors are finally added to another vector that now represents the event-database. This vector is finally returned by the method.

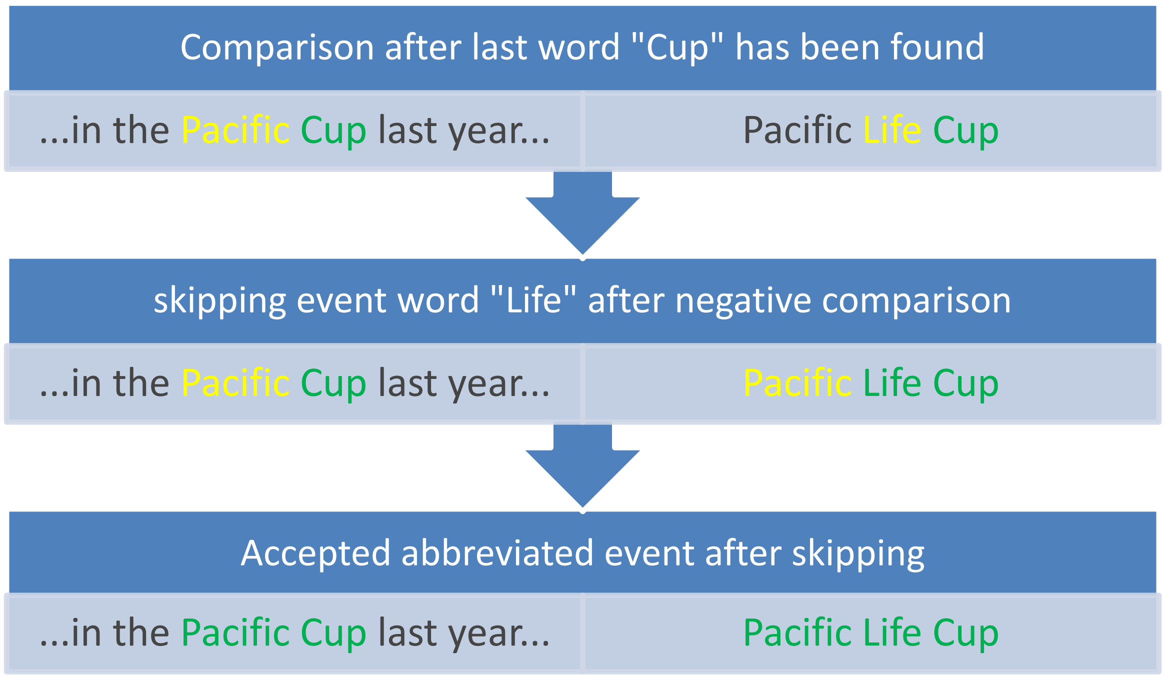
As a last step of preparation, the event module reads in another event-file that contains the earlier mentioned "special events". The method used therefore is the "readSpecialEventsFile(String filename)". It combines the functionality of reading the file from the file system and converting its contents to a usable format, which is again a vector. The manner of how this method works is straight forward to the previous described method. The returned vector includes single strings that represent a special event word.

The method "compareToDatabase(Vector input, Vector database, Vector specials)" performs the actual act of parsing and recognizing the event entities in the text. It therefore incrementally checks each word of the input vector and compares it to its two databases (events and specials) in different ways. To understand the following steps it is crucial to know, that all comparison is based on the last word of an event. The comparison is divided into 3 basic steps. The first step searches for direct event matches from the event dictionary. It therefore compares the current word with the last word of each event-vector in the event database. If there is a match, the previous word will be compared until a complete match is found. If a match is found, the algorithm stops searching and moves on the next word of the input vector, otherwise the algorithm will continue searching for extended matches. This means, the algorithm did find a matching last word in the dictionary, but the event did not match the words in front of the current word in the text. The algorithm will therefore now skip the word in front of the current word in the text and try to match the next previous text word with the event word. It this will match, the event is recognized as an extended event. The algorithm will skip up to 5 words in order to search for extended events in the text. The only requirement for skipping a word is, that it has to be a "NNP" (Proper Noun). Therefore it is guaranteed, that not two events are mixed into a single one.


Comparison after last word match
Anna has won the Adelaide Women's International
Adelaide International
Skipping NNP in text after negative match
Anna has won the Adelaide Women's International
Adelaide International
Accepted event after skipping
Anna has won the Adelaide Women's International
Adeleide International

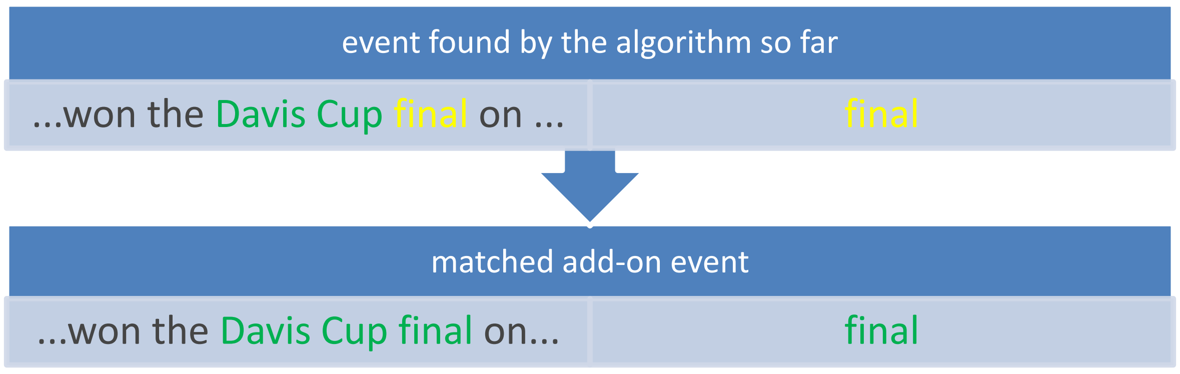
The graphic displays the way extended events are found. The yellow words are currently compared after a matching last word has been found. The right side represents the event in the dictionary, the left represents the news text. After skipping the word "Women's" (must be NNP), the algorithm can match “Adelaide” with the dictionary and therefore accept the whole term “Adeleide Women’s International” as extended event. If this first parsing step will not find a match in the dictionary, the second step of parsing is performed.

In the second parsing step, the algorithm checks for abbreviated events and special event add-ons. The abbreviation algorithm works similar to the way extended events are found. The difference is that this time words in the current event-vector are being skipped. The graphic depicts the way, the abbreviation algorithm will skip event words to find events.


Comparison after last word "Cup" has been found
...in the Pacific Cup last year...
Pacific Life Cup
skipping event word "Life" after negative comparison
...in the Pacific Cup last year...
Pacific Life Cup
Accepted abbreviated event after skipping
...in the Pacific Cup last year...
Pacific Life Cup

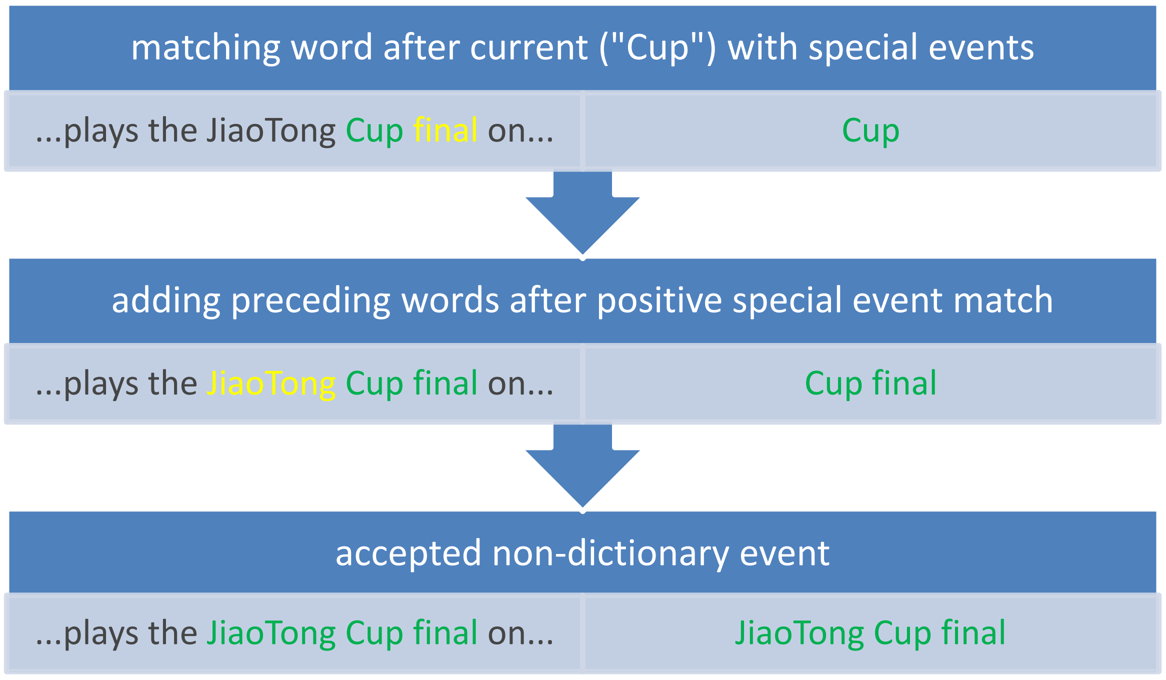
The right side represents the database event, while the left side shows the part of text currently worked on. The green words have been successfully matched, while the yellow words are the words currently being matched. The word “Life” is skipped after a negative match with the word “Pacific” in the text. Since the next word in the event matches the current word in the text, the abbreviated event “Pacific Cup” is accepted as abbreviated event by the parser.

Next, the parser will check, whether the found event could be an special event. It therefore checks, whether the word after the current found event is a proper noun (NNP) and (if so) compare it to its list of special events. If the next word happens to be one of those, it is added to the current event and recognized as one event. The parser can therefore find special events as shown in the graphic below.


event found by the algorithm so far
...won the Davis Cup final on ...
final
matched add-on event
...won the Davis Cup final on...
final

The right side represents the special event database word, the left side the current sample of the text. After the event "Davis Cup" has been discovered, the word after is examined for a possible special event since it also is a proper noun (NNP). The word "final" is therefore added to the event and accepted as extended special event "Davis Cup final".

In the third and final step of parsing, the algorithm will search for events that are not found within the database, based on the special events database. It will therefore further examine words that precede words found in the special events database. To make this step a little clearer, it should be said once more that at this time of parsing, no event match has been found for the current word of the text. So if the current word is not matching with any method of events, but still a proper noun, the algorithm will check its successor and determine, whether it could be a special event. If so, the word is probably also an event. Next, the preceding words are examined. If they are proper nouns, they will be added to belong to the found non-dictionary event. This will add all preceding proper nouns to belong to the event.


matching word after current ("Cup") with special events
...plays the JiaoTong Cup final on...
Cup
adding preceding words after positive special event match
...plays the JiaoTong Cup final on...
Cup final
accepted non-dictionary event
...plays the JiaoTong Cup final on...
JiaoTong Cup final

The graphic explains the non-dictionary algorithm. The right side represents the current found word/event. On the left, the currently examined text is depicted. The yellow word is currently examined, while the green has been accepted as possible match. Since the current word “Cup” is a proper noun that doesn’t belong to any event in the event database (since JiaoTong Cup is not in the database), it is examined to be a non-dictionary event. Therefore the word after will be compared to the special event database. “final” will be found, so it is added to be a non-dictionary event. Now all proper nouns in front of the current text word “Cup” will also be added to belong to the event. Therefore “JiaoTong” will also be added to form the final match of “JiaoTong Cup final”. This is the last step of parsing for events in the algorithm. Each event that is found to be an event will be added to a vector that holds event items in form of a String[] array. Each string array includes three strings. The first string is the complete found event string. The second string is a converted integer that indicates the number of words that form the complete event, eg. The event “Davis Cup quarterfinals” would have a number of 3 words. The third string is a converted integer value that indicates the position of the first word of the event in the text, e.g. if the whole text starts with “Yesterday the official Davis Cup has begun…”, the value for the “Davis Cup” event would be 4. The two numerical values help the program later to determine where in the text the events have been found, so they can be highlighted in the text correctly.

The last method in the module is also a deprecated testing method. The “writeOutput()” method uses the returned vector of the “compareToDatabase()” method and the complete string holding the original news text, to output a string that is includes the entire news text with the found events tagged at the right position in the wanted format, e.g. “…won the [EN Davis Cup] again…”.

The Event module also makes use of the “removePunctuation(String word)” method, which will remove any punctuation at the beginning or end of a word in the parsed vector in order to being able to compare the word to the dictionary events. For further details on this method, refer to the annotated source code of the project.

ii. Working Sequence

The sequence of calling the different methods in the module is of great importance, since the outputs of the methods are again needed in other methods. Therefore the sequence of calling the methods is defined in the method “parse()” which receives the string containing the tagged news text. This will then be tokenized by “tokenizeInputFile()”. The returned vector will then be used in the “compareToDatabase()” method, after the methods for reading and tokenizing the database files have been executed to return vectors holding the event and special event data. These three vectors (input data, event data and special event data) are then passed to “compareToDatabase()”. This method again returns a vector as explained in the section above, which is the final return value of the parse() method (and implicitly the event module).

d. Person Names

For the person names, two implementations are chosen. A dictionary and a rulebased approach. The rules can be deactivated by via triggering a simple Boolean within the class.

i. Dictionary recognition

The main recognition of names is done with a dictionary. The dictionary file consists of around 1000 most famous male tennis player names and another 1000 most famous female tennis players. This list of names has been taken from current world rankings and a global list of important tennis players found on Wikipedia.org (6). As the headlines of Tennis.com only report about the most famous events in the tennis domain, this dictionary approach already recognizes around 90% of all names in the headlines. For the recognition, a simple textfile containing the player names is used as the dictionary. This textfile is read first and its contents is saved within a vector of strings, first and lastnames separately from each other. Like this, first or last names of players will be recognized independently in the text and not only in a combined context. Then the dictionary algorithm is like:

- Take actual NNP-word
- Circle through list of dictionary words
- Break if found and return Boolean true
- If never found, return false

ii. Recognition based on rules

Additionally, some rules where implemented according to the following algorithm:

- If current word is NNP, check previous word for Grammars like JJ, NN, VBZ, CC, NNP
- If true, increase possibility counter
  - If CC (a word like "and") test if 2nd word before is NNP as well to increase counter again
- Then check following words of current word for same grammars
- If true, increase possibility counter
  - If CC (a word like "and") test if 2nd word after current is NNP as well to increase counter again
- Read counter and mark words with value above threshold as recognized names
- Repeat algorithm again (already marked words will cause higher weight for e.g. NNPs following them or in constellations like: "*marked name – and – NNP*"

Rule Description:

If a NNP is followed by certain words like a JJ, NN or VBZ (e.g.: *Guccione/NNP qualifies/VBZ for…)*, the NNP is considered to have a good possibility to be a Name. Each NNP in the text is examined in the same way. The following two words and the previous two words of the current NNP are checked according to those rules and a counter *isName* is increased each time such a constellation is true. Other factors for increasing this value are a NNP followed by another NNP as this could be a match for a first- and last name, a NNP followed by *"and"* and then another NNP, as this could be an indication for a list of names etc. Higher points are as well achieved if within such a constellation a name from the dictionary is found. If e.g. the first name is in the dictionary and then followed by *"and"* and another NNP, this NNP most likely is a name too.

Another factor is if one NNP is found in the dictionary and the previous word or the following word is a NNP too, it may be a name but e.g. another first name like e.g. a relative to a famous player (like *"Barbara Becker and Boris Becker celebrated his triumph…"*). After these rules, the value of the counter is checked and if a certain threshold is reached, the word is marked to be a name (through tagging it with an X"). Then, the same algorithm is repeated again as now previously recognized names could help to identify previously not recognized names through adopting again the rules.

iii. Result

This algorithm proved to perform pretty good but had one big drawback: As the news page sometimes was written in a not so common way, some errors occurred as well. Sometimes the news where: "*Spain won the semifinals against….",* were Spain then was recognized as a Player Name. Normally it would not be Spain who won the game but a player from Spain in such a constellation, but written like that, Spain will be wrongly identified. This problem can not be solved by simple rule sets, but maybe through blacklists of such constellations or other learning methods in the future.

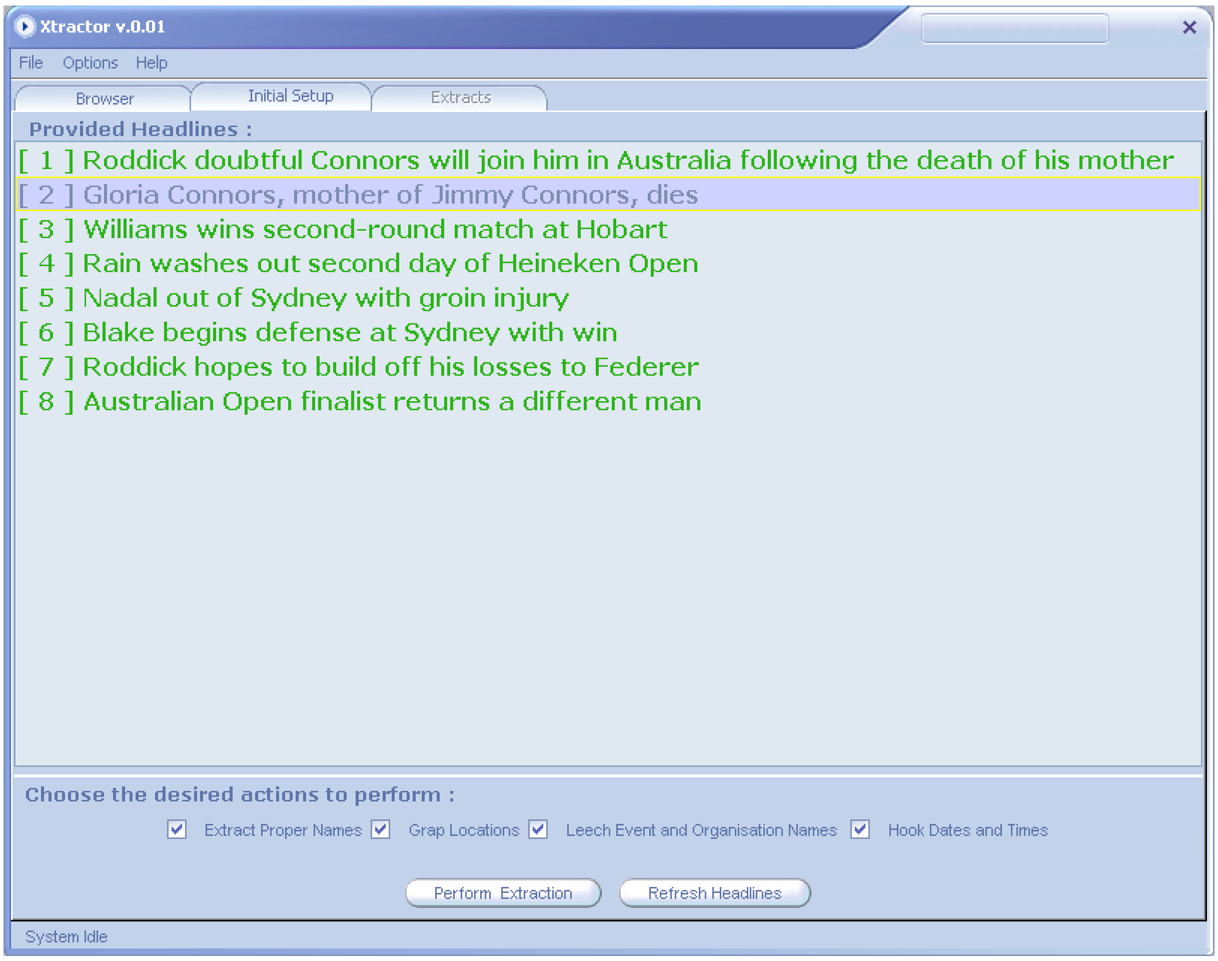
5) User Handbook

After starting the Xtractor program the user can choose between an online and offline modus.

a. Online Mode

The online mode will establish a direct connection to the website (www.tennis.com) and read the html. The website will then be visible in the HTML tab like in a regular browser. The tab besides it can be clicked to view the html-source as plain text.

To choose which news article should be extracted, click on the "Initial Setup" tab to view the headlines that have been extracted automatically from the front page of the website. Click on the headline of your choice to select the corresponding news article. Below the list of headlines, choose the checkbox of each extraction-task you wish to be performed. These are the four named entity recognition tasks for date & time, locations, events and player names. After you have made your selection, hit the "Perform Extraction" button on the bottom of the window to initiate the extraction task.



Provided Headlines :

[1] Roddick doubtful Connors will join him in Australia following the death of his mother
[2] Gloria Connors, mother of Jimmy Connors, dies
[3] Williams wins second-round match at Hobart
[4] Rain washes out second day of Heineken Open
[5] Nadal out of Sydney with groin injury
[6] Blake begins defense at Sydney with win
[7] Roddick hopes to build off his losses to Federer
[8] Australian Open finalist returns a different man

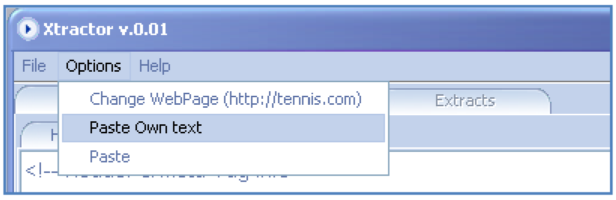
Choose the desired actions to perform :

[✓] Extract Proper Names [✓] Grap Locations [✓] Leech Event and Organisation Names [✓] Hook Dates and Times

After extraction has finished, the results will be displayed in new tabs, one result tab for each module containing the recognized named entities.

b. Offline Mode

The offline mode is chosen, if there is no internet connection available or to parse a text that is not from the tennis.com website. To insert an own text of your choice, click on the menu item “Options” and choose “paste own text”.



In the following opened window, paste your own text and click on “Parse Text” to initiate the parsing. After parsing has finished, click on the new result tabs of the main program window to view the results of parsing.

6) Conclusion and Comment

To conclude the project, we summarize the work and research done by us. After we have found it rather difficult to decide which method of Information Extraction to choose for our specific task, we have decided to work with hybrid dictionary/rules approaches on an easily extendable modular architecture, since we found single approaches to be not efficient enough. Stepping deeper into the subject, the task proved to be not as simple as we first thought. With the current version of our project we achieved a usable information extraction application that is working good- but without perfect accuracy for the selected domain. We learned that no rule would ever achieve 100% correct results and no dictionary would ever be 100% complete. Therefore, our program has a potential to be improved in the future.

To improve our program, we have several ideas and approaches that could be conquered in future work. One of the ideas was the extension of communication in between the different parsing modules. The program could be extended that the location module would help the person module to achieve more accurate results by eliminating ambiguous matches that could possibly be recognized as person entities, even though they are actually locations. E.g. a rule that determines person entities by the pattern “X wins against Y.”, where X and Y could be either persons or locations/countries.

Another idea was to remove the constraint of only being able to parse news articles of the website tennis.com and implement a more general html-parser which is able to parse texts from any arbitrary tennis website. Also we had the idea of a module to automatically retrieve and update the dictionaries used by the different parsing modules with current player names, tennis events and locations to further increase the efficiency of the program.

Bibliography

1. Tennis.com. *The official site of TENNIS Magazine.* [Online] [Cited: January 13, 2007.] http://www.tennis.com.

2. **Microsystems, Sun.** Java.com. [Online] Sun. [Cited: January 13, 2007.] http://www.java.com.

3. Eclipse.org home. [Online] The Eclipse Foundation, 2006. [Cited: December 19, 2006.] http://www.eclipse.org/.

4. The Most Intelligent Java IDE. [Online] JetBRAINS. [Cited: January 13, 2007.] http://www.jetbrains.com/idea/.

5. **Group, The Stanford Natural Language Processing.** The Stanford NLP. *The Stanford Natural Language Processing Group.* [Online] Stanford University, 5 21, 2006. [Cited: December 19, 2006.] http://nlp.stanford.edu/software/tagger.shtml.

6. **Wikipedia.org.** World Tennis Plaer List. *Wikipedia, the free encyclopedia.* [Online] [Cited: December 5, 2006.] http://en.wikipedia.org/.