PTkChA
A Perl/Tk Chunk Annotator

Edgar Gonzàlez i Pellicer

Contents

1 What is PTkChA?

PTkChA is a simple utility for annotating chunks in text files. It offers a GUI to aid in the task of annotating non-overlapping fragments of text, as well as support for Plugins.

It has been written using Perl/Tk by Edgar Gonzàlez i Pellicer[1] at the TALP Research Center in Barcelona.

1.1 License

PTkChA is Free Software, and it is released under the General Public License (GPL). A copy of the GPL is bundled with the PTkChA source code.

2 Installation

2.1 Requirements

PTkChA requires the following components installed in your system. We give the versions we have used, other versions will probably work.

- **Perl-5.8.8**, `http://www.perl.com`.
- **Expat-1.95.8**, `http://expat.sourceforge.net/`.
- **Tk-804.027** Perl module, `http://search.cpan.org`.
- **XML-Parser-2.34** Perl module, `http://search.cpan.org`.

2.2 Installation

Download the last version (currently 2.8.1) from `http://www.lsi.upc.edu/~egonzalez/ptkcha.html`, and uncompress the tar file you will find there in a directory of your choice. **PTkChA** only needs permission to write its configuration file, `.ptkcha.xml`, in your home directory.

2.3 Starting the program

Issue the command:

```
> perl ptkcha.pl
```

from the directory where you installed it. The standard output of the process acts as an error log, so you might find it interesting to keep it.

```
> perl ptkcha.pl | tee ptkcha.log
```

3 Concepts

The main concepts in **PTkChA** appear in Figure 1.

Work is arranged in **Projects**. A **Project** is defined by an **Input Directory**, an **Extension** to select the files in the input directory we are interested in, a **Filter** for importing the files, an **Output Directory** and a **Marking**, defined in a file.

The philosophy of work is that the files in the **Input Directory** with the specified **Extension** have to be annotated. **PTkChA** offers an interface which allows to select one of the files, which is preprocessed through an input **Filter**. The file is then available for annotation, and the kind of annotation the interface allows is defined in the **Marking**. Lastly, the results are output to the

[1] `http://www.lsi.upc.edu/~egonzalez`

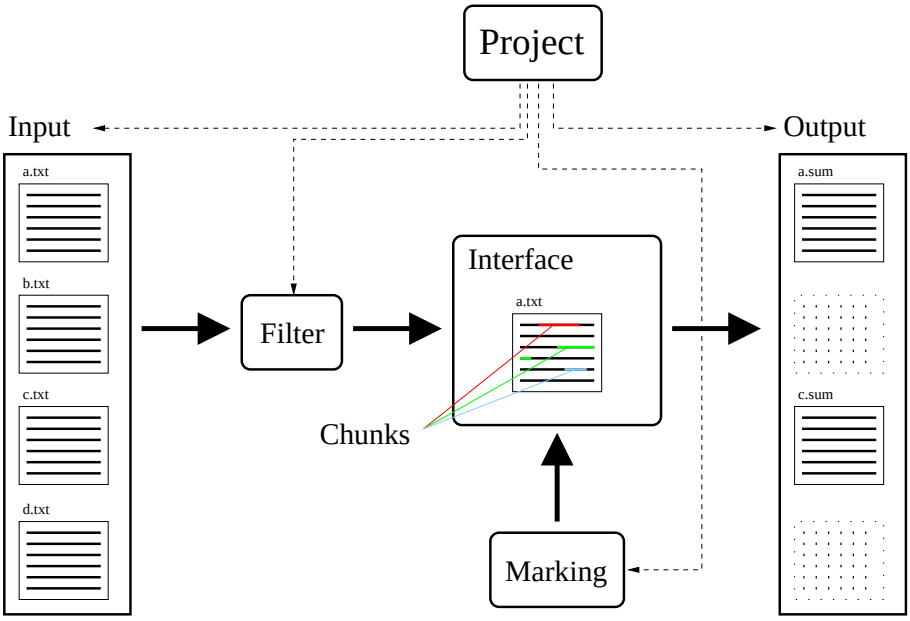

Figure 1: Main concepts in PTkChA

Output Directory, in an XML file with the same name as the input one, but with extension **.sum**[2] (see Section 6).

Further accesses to this file will not take the raw input file from the **Input Directory**, but rather will take the already annotated version from the **Output Directory**.

The annotation process consists in the identification of **Chunks**, non-overlapping segments of the input text, as well as the definition of their **Attributes** and the **Relations** between them (see Figure 2). The attributes and relations that may be defined are specified in the **Marking** (see Section 4). Each chunk has a unique identifier (ID), and if clusters are defined, it also has a cluster identifier (SUBST, see Section 4).

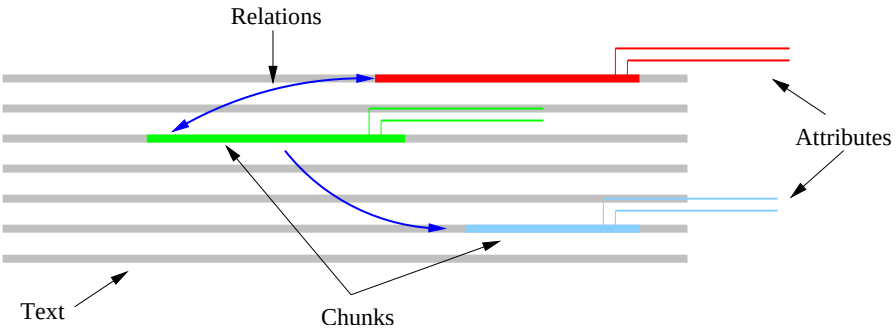

Figure 2: Chunks, Attributes and Relations

[2]The reason for the **.sum** extension instead of **.xml** is historical: the program was initially created only for annotating summaries, and hence the **.sum** extension.

4 Marking Files

A sample of a marking file is that of Figure 3.

```
<marking element="ne">
 <attribute name="class">
  <value v="PERSON" />
  <value v="ORGANIZATION" />
  <value v="LOCATION" />
  <value v="OTHERS" />
 </attribute>
 <attribute name="multiword">
  <value v="No" />
  <value v="Yes" />
 </attribute>
 <relation name="coref" stereo="bi" />
 <plugin name="NE" class="PluginNE"
   file="Plugins::PluginNE::PluginNE" />
 <clustered />
 <extra element="DOC"/>
</marking>
```

Figure 3: Sample Marking File

The main element is the **marking** element. Its attribute **element** simply gives the name of the tag that will be used in the output files of the project to delimit the chunks. So, in this case, the output files will contain `<ne>...</ne>` pairs. Inside the **marking** element come the other elements:

- **attribute** elements define attributes the chunks will have. Inside the **attribute** element come a series of **value** elements which define the possible values for it.
- **relation** elements define possible relations between chunks. As well as its **name**, **relation** defines the stereotype (**stereo**) of the relation, which may be `uni` or `bi`, depending whether the relation is unidirectional or bidirectional. The difference between them is that in the latter case, the relation is assumed to be reflexive, which means that the creation of a relation from a source chunk to a target implies the relation on the opposite sense; whereas in the case of an unidirectional relation this is not necessarily true.
- **plugin** elements define which plugins will be active when working with this marking. For every plugin, we must define its **name**, which identifies it in the PLUGINS menu (see Section 5.4.5), as well as the name of the **class** which implements it and the **file** where it is located. The files are searched in the default Perl directories, as well as in those listed in the global option **Include Directories** (see Section 5.4.1). See section 7 for more details about plugins.
- If the **clustered** element appears, then an equivalence relationship can be defined between the chunks in every document. In the output file, equivalent chunks will share a common SUBST.
- **extra** elements give other XML tags that might be present in the input files, and which should be preserved in the output files. If the `<` character is found in an input file, it will be substituted by its XML escaped equivalent `&x3c;`, unless it is part of one of an XML tag. The list of **extra** elements given is used to determine what actually is an XML tag.

Some sample markings are included with PTkChA, in folder `markings/`.

5 Usage

5.1 Main window

The main window is divided in 4 regions (see figure 4).

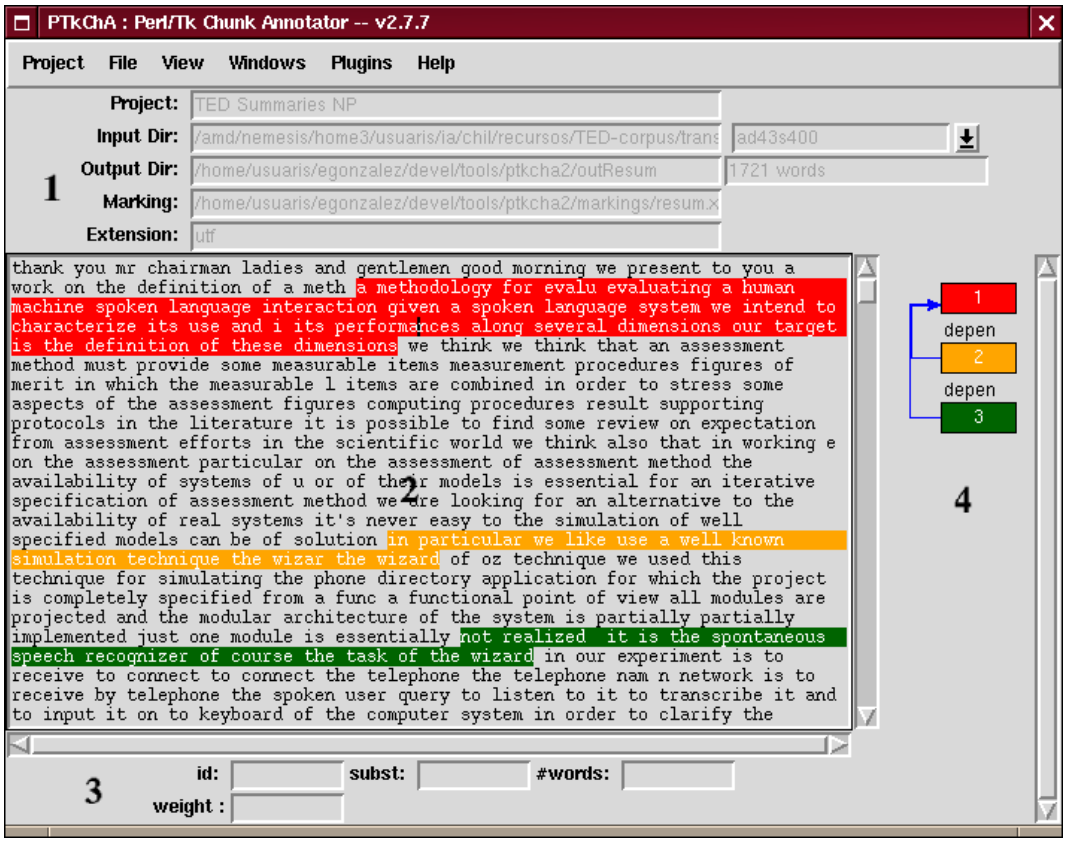

Figure 4: Main window

1. **Project Information** Here we can see the information about the current project, as well as select, by the combo box in the right, one of the files to annotate it. The text under the combo shows the number of words in the document[3].

2. **Text Area** Here we can see the text of the currently selected file, and perform several operations of it.

   - To create new chunks, select a region of text in the usual way, and then either press the RETURN key or click the right mouse button. If the option **Selection Expansion** is set (see section 5.4.1) the selection is expanded so it does not include extra whitespace beyond the first and last word of the chunk, and so it does not finish in the middle of a word.
   - To reduce an existing chunk, select a region of text strictly included in the chunk and do as in the previous case.

[3]This number of words is just an informative measure. It does not intend to give the precise number, but it simply counts elements separated by whitespace

- To expand an existing chunk, select a region of text that overlaps but is not strictly included in the chunk, and do as in the previous case. The chunk will grow to the union of both regions.
- To see information about a chunk, move the pointer over it and the information will appear on the chunk information region (identified as 3 in Figure 4).
- To see the relations of a chunk, click the left button on it, and the information will be shown on the relation information region (identified as 4).
- To edit the chunk, click the right button on it, and a pop-up will appear, allowing us to:
  - Change the values of its attributes.
  - Add a relation to another chunk. After selecting this option, the cursor will change to a cross and we will be able to select the target chunk with the left button. This can be canceled with the ESCAPE key or by pressing the right button.
  - Delete the chunk. This will also delete all relations having the chunk as source or target.

3. **Chunk Information** This region shows the ID and SUBST of the chunk currently under the cursor, as well as the values of its attributes and its length in words.

4. **Relation Information** This region shows the relations in which the current chunk is involved, either as source or as target. Clicking the left button on the boxes depicting the chunks related, the text will scroll to locate them. Clicking the right button, we will have access to a popup giving us the chance to delete the relation.

For users with single-button mice, those operations requiring the right button may also be performed by holding the SHIFT key while clicking the left button.

5.2 Project Manager

The project manager window lists the available projects, and allows its creation, modification, deletion and cloning, as well as its opening as current project in the main window. An image of the project manager window is shown in Figure 5. The information for each project includes its status (which can be *Active* for the currently selected project, *OK* for projects that have no errors and *Error* for those projects that have errors which prevent its use), its name and the name of its marking.

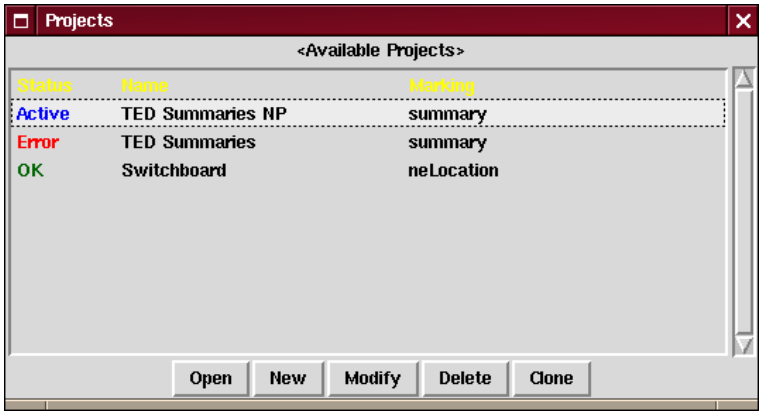

Figure 5: Project manager window

Project maintenance tasks are invoiced using the buttons in the window. Selection of a project is accomplished by clicking on it:

Open Opens the project in the main window.

New Creates a new project. A project window (see Section 5.3) is opened to get the information about the project.

Modify Opens a project window which allows the modification of the parameters of the selected project. The active project cannot be modified.

Delete Deletes the selected project. The active project cannot be deleted.

Clone Clones the selected project, this is, creates a new project with the same parameters.

5.3 Project Window

To create a new project or modify an existing one, a special purpose window is opened. It allows the edition of all the parameters defining a project: name, input and output directories, marking, filter and extension (see Section 3). An image of it can be seen in Figure 6

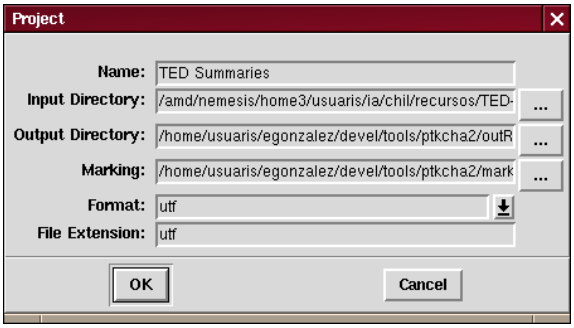

Figure 6: Project window

If there is an error in any of the parameters, its value will be highlighted in red. When the mouse pointer is on the value, an explanation of the error will appear on the top of the window (Figure 7).

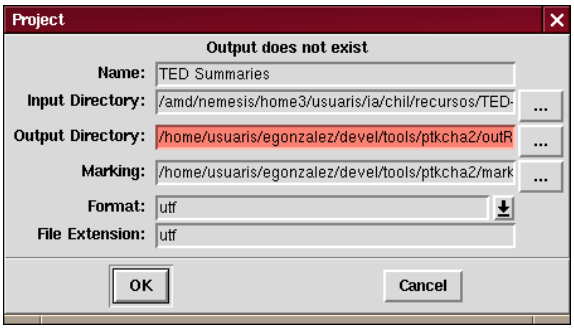

Figure 7: Project window showing an error in the parameters

5.4 Menus

5.4.1 Project

Manage... Opens the project manager window (Section 5.2).

Options... Allows the changing of global options:

Forced Writing If a file has not been modified from its state in the input directory, by default its counterpart in the output directory is not written. By activating this option we can force the output file to be written.

Selection Expansion Expands the selection when creating new chunks, as explained in Section 5.1.

Include Directories Gives a list of additional directories to look for the source files of plugins and filters (see sections 7 and 8).

Quit Quits the program.

5.4.2 File

Save Usually, changes are saved automatically whenever the current file is changed: another one is selected, the project is changed, the program is closed... With this command, we can force the file to be written immediately.

Revert Allows us to lose the unsaved changes and revert to the last saved version of the file.

Initial Version Allows us to lose all changes and come back to the initial version of the file, that of the input directory.

Default Values... Allows the changing of the default values new chunks will have for their attributes.

5.4.3 View

On the Text Area, the color of every chunk may be just an identifier of the chunk or of the cluster it belongs to (in case of a clustered marking), or may also be a function of the value of an attribute. With this menu we can change the current view mode. The correspondence between colors and values can be examined using the Chart window (Section 5.4.4).

5.4.4 Windows

Chart Opens a Chart window. The Chart window accomplishes a double task: it gives the values of the attributes for every color in the current view mode (see Section 5.4.3), and it also gives the number of words in chunks with every attribute value, as well as the fraction of the total of words in the document they represent. The discriminant attribute in the Chart window depends on the current view mode, selected in the VIEW menu. A sample chart window is shown in Figure 8

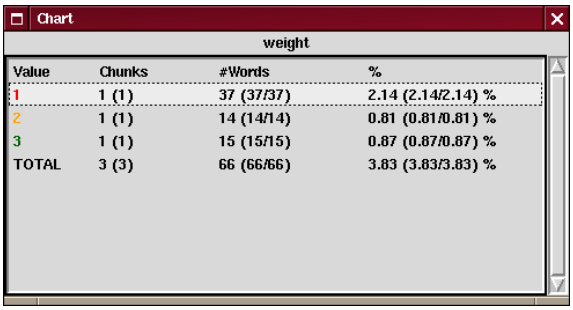

Figure 8: Chart window

5.4.5 Plugins

This menu is initially empty and, when a project is opened, it is populated with the menus provided by the plugins defined in the project's marking.

5.4.6 Help

About... Shows a small window with information about the program.

6 Output Files

The output files are *headless* XML files, in the sense that there is no element that embraces the whole file. This makes them not strictly XML-compliant, yet suitable for most practical purposes. If XML-compliance is an issue, including the file contents in a `<document>..</document>` pair or similar is enough. A sample of the contents of an output file is in Figure 9.

```
thank you mr chairman ladies and gentlemen good morning we present to
you a work on the definition of a meth <summary id="1" subst="1"
weight="1">a methodology for evalu evaluating a human machine spoken
language interaction given a spoken language system we intend to
characterize its use and i its performances along several dimensions
our target is the definition of these dimensions</summary> we think we
...
looking for an alternative to the availability of real systems it's
never easy to the simulation of well specified models can be of
solution <summary id="2" subst="2" weight="1"><rels><rel type="depen"
target="1" /></rels>in particular we like use a well known simulation
technique the wizar the wizard</summary> of oz technique we used this
technique for simulating the phone directory application for this
```

Figure 9: Sample Output File

The file contains the imported text from the input directory. Each annotated chunk is enclosed inside an XML element. The type of element determined by the **element** attribute in the marking file (see Section 4). In the sample case, it is `summary`. The attributes of the element are its unique ID, its SUBST or cluster ID (only if the marking is **clustered**) and the values for all attributes of the chunk. In the sample case, the only attribute defined in the marking is **weight**.

If some relation has the chunk as source, immediately after the opening tag of the chunk comes a **rels** element, which consists of a list of **rel** elements describing the **type** and the **target** of the relation. The target chunk is identified by its unique ID. If there are no relations coming out of the chunk, the **rels** element is omitted.

7 Plugins

PTkChA offers support for runtime-loading plugins, which offer additional functionalities depending on the type of marking we are using.

A sample of the kind of functionalities plugins may provide appears in Sections A.3 and A.4.

A description of the interface to be followed by plugins will appear in a future version of the documentation.

```
<marking element="phrase">
 <attribute name="type">
  <value v="NP"/>
  <value v="PP"/>
 </attribute>
 <attribute name="prep">
  <value v="of"/>
  <value v="in"/>
  <value v="to"/>
 </attribute>
</marking>
```

```
The   DT  B-NP
cat   NN  I-NP
eats  VBZ O
tons  NNS B-NP
of    IN  B-PP-of
fish  NN  I-PP
.     .   O
```

```
<phrase type="NP">The cat</phrase> eats <phrase type="NP">tons</phrase>
<phrase type="PP" prep="of">of fish</phrase>.
```

Figure 10: Marking, BIO input, and equivalent XML file.

8 Filters

The filters allow the use of several inputs formats for the projects in a flexible way. New filters can be added by implementing them as Perl modules and, by now, modifying the user configuration file. However, **PTkChA** comes with a standard set of them.

A description of the interface to be followed by filters will appear in a future version of the documentation.

8.1 Standard Filters

bio It imports files in BIO labeling format (such as the one used usually in Named Entity Recognition and Classification). The expected format of the input is a file with information in columns. The first one of the columns is taken to be the word form, and the last one the BIO label. The BIO label can contain extra information separated by dashes (-), which are taken to be the attributes of the chunk, as given in the marking file. So, given the marking and input files in the top of figure 10, the equivalent XML file would be the one in the bottom of the figure.

txt The simplest filter, it performs no conversion on the input files.

utf It reads files in Universal Transcript Format and leaves only the spoken words information.

utf_np It also reads Universal Transcript Format files, but in addition, removes capitalization and punctuation information.

xml Reads XML files, trying to preserve the XML tags present in them. If there already is some information about the marked chunk in the file, it is imported.

yam It reads the output of the YamCha [KM01] chunker when applied to Named Entity Recognition and Classification with the MUC classes (Person, Organization, Location and Others).

A Summary Annotation Framework

A.1 CHIL Summary Creation Methodology

The [FGT05] CHIL project deliverable defines a methodology for the creation of summaries.

From the raw text, human annotators start by identifying relevant text fragments (chunks) and giving them a weight (1, 2 or 3) according to their relevance. Also, chunks considered equivalent have to be identified, as well as chunks whose presence in a summary is dependent on the presence of another one to allow understanding.

Lastly, summaries of different sizes have to be built by selecting a number of chunks from the relevant set.

PTkChA includes a marking and a plugin for summary chunk annotation and summary creation following this methodology.

In addition, to ease the creation of summaries of automatical transcripts, [FGT05] contemplates the transferal of the chunks selected for the construction of summaries on manual transcripts to automatical ones. This implies finding, for each chunk on the manual transcripts, the equivalent chunk in automatical ones, taking into account the recognition errors.

This marking also includes a plugin which finds the alignment of the words in two texts minimizing word error rate and transfers the chunks annotated in one of them to the other, giving complete support for the CHIL methodology for the creation of summaries.

A.2 Summary Marking

The summary marking defines:

weight An attribute which gives the relevance of the chunk, in a range of three values (1, 2, 3).

depen An unidirectional relation of dependency between a chunk and another chunk which should be present in the summary if the former is.

clustered The chunks are clustered. Those chunks belonging to the same cluster are equivalent. Hence, only one of the chunks in a cluster needs to be present in the summary.

A.3 Summary Plugin

The plugin offers two main functionalities: that of generating summaries from the annotated text and that of checking the dependency relations. Both can be accessed through the PLUGINS menu.

A.3.1 Generate Summary

The interface of this plugin functionality is that of figure 11.

The slide bars on the top of the window allow to restrict the size of the summary to a percentage of the words in the document, or to a fixed number of words. In the middle a table with every annotated chunk and information about it is shown. We can also select and unselect a particular chunk to appear in the summary using the check boxes to the left of them. The produced summary is written in the lower part of the window, and its length is in the text entry between the chunk table and the summary.

To the left of it, we have the `Auto-Summary` button. This button allows us to get an automatic base summary from which to start the selection process. This summary is built as follows:

1. All chunks of weight 1 are selected.
2. If we do not reach the desired number of words, all chunks of weight 2 are also selected.
3. If we do not reach the desired number of words, all chunks of weight 3 are also selected.
4. At every step, if a series of chunks are considered to be equivalent, only the first one of them is selected.

After the desired chunks are selected, we may choose a name for the output file and generate it using the `Generate` button. Before the generation, the set of chunks is verified, so no two equivalent chunks be included, nor a dependency relation be broken.

We may dismiss the window using the `Close` button.

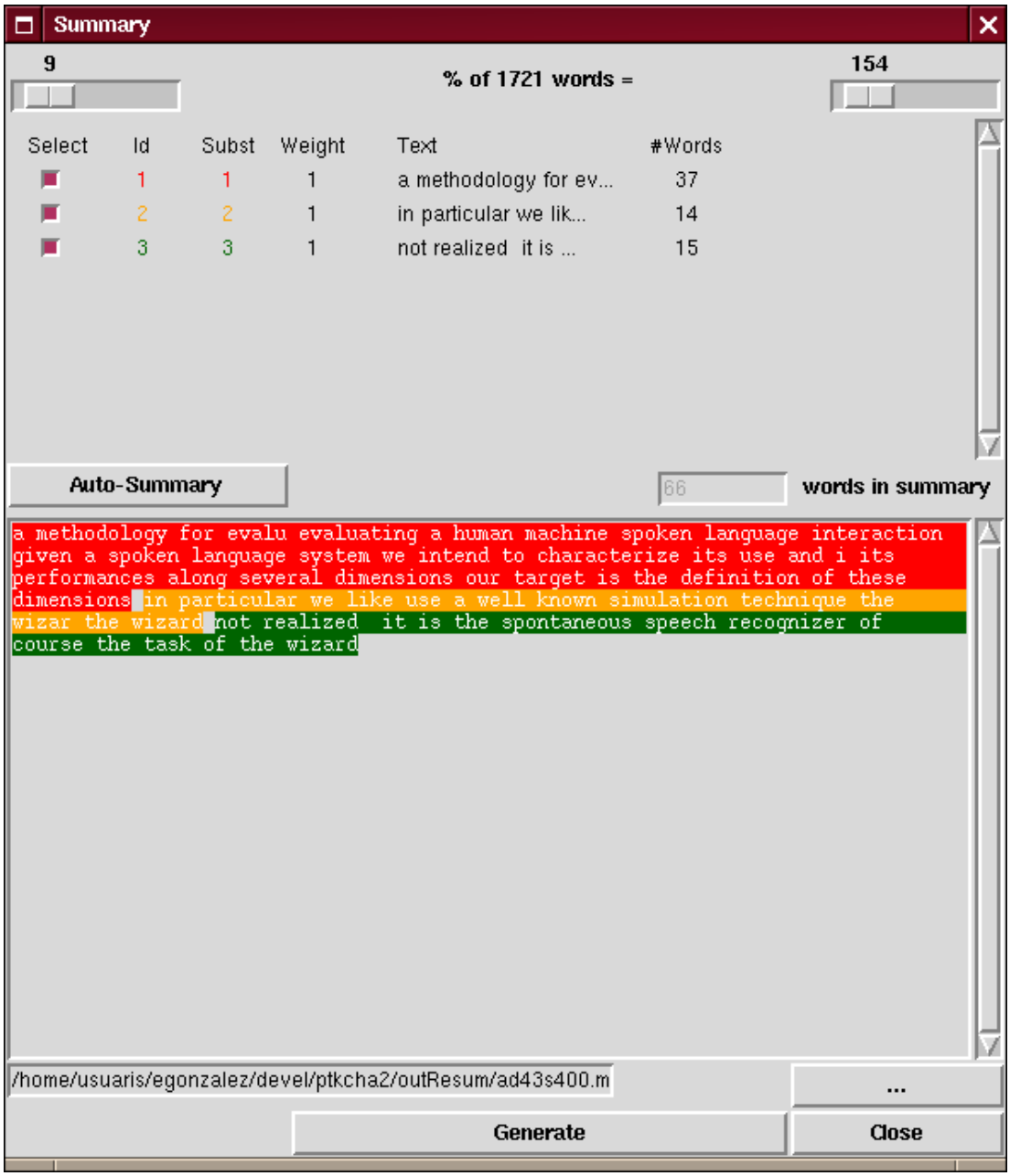

Figure 11: The Generate Summary interface

A.3.2 Check Dependencies

This option just checks the sanity of the `depen` relations: no chunk should depend of another one of lower relevance.

A.4 Alignment Plugin

The plugin allows the transfer of a set of annotations from one text to another, accessed through the PLUGINS menu.

The plugin dialog window (depicted in Figure 12) allows three different kinds of alignment:

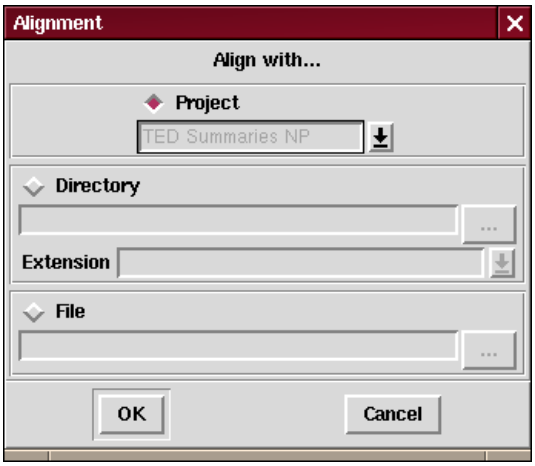

Figure 12: The Align Project interface

Project Each file in the current project is aligned with the file with the same name from the selected project.

Directory Each file in the current project is aligned with the file with the same name and the given extension in the given directory.

File The current file is aligned with the given file.

In all cases, the chunks are transferred **from** the given files or project **to** the current file or project.

References

[FGT05] M. Fuentes, E. Gonzàlez, and J. Turmo. Baseline summarization system for text including speech transcripts. European Project CHIL (IP 506909) Deliverable D5.8, 2005.

[KM01] T. Kudo and Y. Matsumoto. Chunking with support vector machines. In *Proceedings of the NAACL*, 2001.