jTaCo Userguide

Christian Bering

January 19, 2004

1 Introduction

1.1 Is this for you?

The purpose of JTaCo is to help evaluate grammars. The user has to provide

- an *annotated corpus* and
- a *parser* which she can feed with the grammar she wants to have tested against the corpus.

JTaCo extracts the annotation given in the corpus and treats it as the 'true' annotation of the corpus. It then compares this annotation to the one generated by the user's parser (resp. her grammar) when run on the same corpus. JTaCo can output statistical information (precision, recall etc.) as well as detailed occurrence lists of items that were or were not correctly identified in the parse.

1.2 JTaCo Processing Stages Overview

JTaCo is highly modular and works in four separate processing stages. Each of these stages can be configured (and will usually have to be configured) according to the user's needs. The possible settings for each processing stage depend on the concrete stages your JTaCo is set to use (e.g., the processing stage which initiates the parse has to be configured for your specific parser). In general, these four processing stages are:

Reading the Annotated Corpus In the beginning, the user has an annotated corpus against which she wants to test her grammar. For use in the following processing stages, JTaCo extracts from the corpus

- the raw text contained in the corpus and

- its *manual* annotation (also called the *true* annotation).

Settings in this processing stage determine how JTaCo identifies annotation information (*tags*, see also below) in the annotated corpus and in which manner it constructs its internal manual annotation from this information.

Parsing the Extracted Text In this processing stage, JTaCo runs the user's parser on the raw text extracted in the previous stage. The settings needed in this processing stage depend on the specific parser (e.g., which of its components the parser should use etc.). The result of this stage is the annotation of the raw corpus reflecting the user's grammar. This is henceforth also termed the *parsed* annotation.

Comparing the Annotations Now the annotations obtained from the two previous processing stages – i.e., the 'manual' annotation read directly from the corpus, and the 'parsed' annotation obtained through the parser – are compared. For JTaCo, an annotation is a collection of tags, where a tag consists of some linguistic information about a piece of text. Minimally, a tag contains

- some name, e.g., a linguistic label,
- the string to which the label applies,
- token count information about where this string is found in the corpus.

Settings for this processing stage can influence the notion of equality between tags contained in the two annotations. E.g., you might be able to define equivalences over tags which have different labels in the two annotations.

Also, your setup might use tags which incorporate more information, and the equality relation will typically depend on this information (e.g., tags which are augmented with feature structure information are likely to use unification or subsumption).

Generating a Report Finally, you generate a result from the previous comparison. This might either be on screen or in some file (or indeed any other output channel your JTaCo provides). The settings for this processing stage will determine which results (i.e., for which tags) you get to see and/or how the information is formatted.

The rationale behind this architecture is that you can change the settings of any one stage and rerun the process at that stage with the new settings without having to go through the previous process stages, as long as their results are still available. This can be especially useful in the last two stages (Comparison and Report Generation), where you might want to experiment with different settings without repeatedly having to rerun the possibly time-consuming processes of reading the corpus and having it parsed.

Depending on the user's specific annotation format and parser, it might turn out to be necessary to write new modules which can parse that specific annotation or interact with that specific parser. As of June 2003, JTaCo includes support for annotations which satisfy certain regular contraints and for annotations in XML; as far as parsers are concerned, JTaCo features an interface for Sprout using feature structure tags.

2 Using JTaCo With Sprout

After having started Sprout, you can access the JTaCo GUI by selecting JTaCo in the *Tools* menu. In the JTaCo GUI, you can set the options for the different processing stages in the corresponding tabs, and you can start the processing by choosing the corresponding button.

3 Settings

The different processing stages in JTaCo each depend on various settings to work properly. These settings can be adjusted in the corresponding panels of the user interface.

3.1 Annotation Settings

An XMLAnnotationParser reads an annotation from an XML corpus and returns a `SortedTagSet` containing `FeatureTags`. Basically, for each Tag encountered in the input the Parser will return a `FeatureTag`. Each `FeatureTag` contains one `FeatureStructure`, and nested XML-tags in the corpus result in nested FeatureStructures in the FeatureTag. This behaviour can be adjusted through a couple of options which are explained below.

Note the difference between the `FeatureTag` constructed from the XML-tags in the annotation and the `FeatureStructure` which is embedded *in* the `FeatureTag`. The `FeatureStructure` and the span of the `FeatureTag` (i.e., its character count start and end) are minimally used to determine equality,

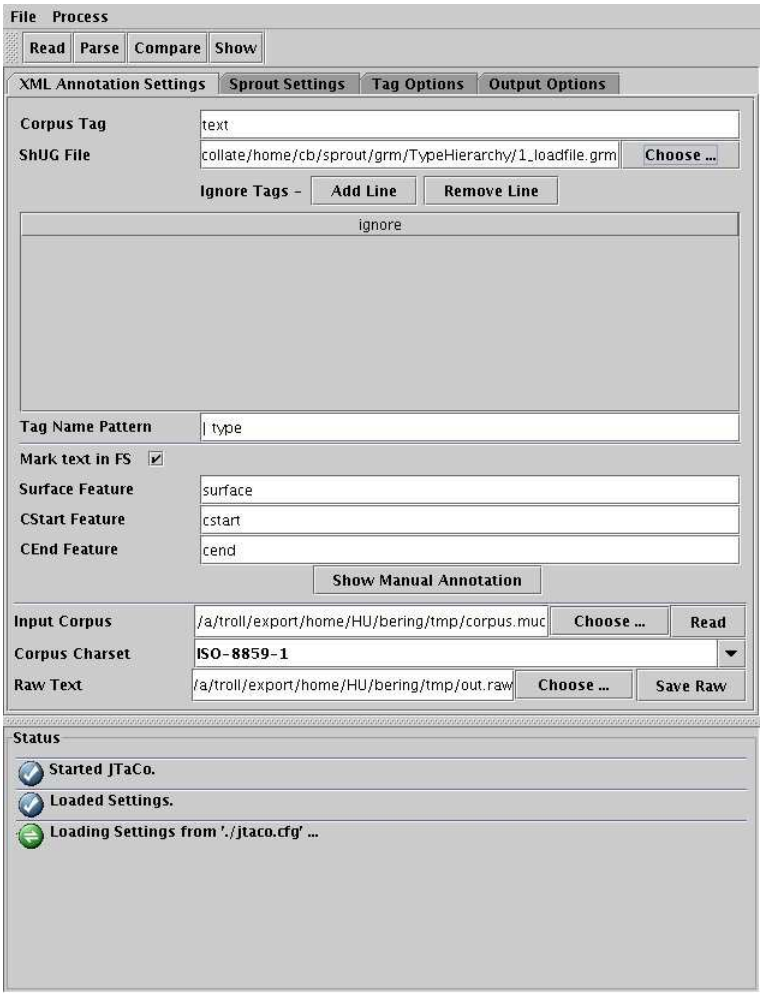

Figure 1: Example annotation settings as they would be seen in the graphical user interface when running JTaCo with Sprout.

the name of the `FeatureTag` might not be.[1] The name of the `FeatureTag`, however, is used to count hits, i.e., it determines the statistics, and it might be relevant for other comparison options. See also the option *Tag Name Pattern* below.

3.2 Options for the XML Annotation Parser

The options that influence the annotation parse are:

Corpus Tag An optional String value which specifies that the Parser should ignore any contents outside of this tag. If given, the Parser will will only consider the text within occurrences of this tag to be the input text, and will construct its annotation only from tags encountered within the scopes of this tag. If this value is not specified, the whole input is treated as being annotated text.

Ignore Tags An optional set of tags which the Parser is not to use for its FeatureStructures. The Parser will still use the text within the scope of the tag, but the tag itself will not constitute a Feature.

Text in FS A Boolean option which tells whether the Extractor should include textual information in the FeatureStructures, recursively. Iff this is set to true, every FeatureStructure gets three additional Features which denote the surface text in this tag, its character count beginning and end. The names of these features can be set by the three option values *Surface*, *CStart*, and *CEnd*. Iff this option is set to false, the textual information will only be included in the `FeatureTag` and not in the `FeatureStructure` contained in the tag.

Surface Tag Only relevant if *Text in FS* is set to true. Then, the String value of this option denotes the feature key to which the value is the raw text marked by the corresponding feature structure. This defaults to `surface`.

CStart Tag Analogous to *Surface*, but its feature gives the character count start of the tagged text. Note that this value is inclusive. The name defaults to `cstart`.

CEnd Tag Analogous to *Surface*, but its feature gives the character count end of the tagged text. This value is exclusive, so the tagged text ends

[1] By default, the name is not used.

with the character at position `cend-1`, as is usual with Java String indices. This defaults to `cend`.

Tag Name Pattern A String which denotes how to build the names for the FeatureTags extracted from the annotation.

This has to be a white-separated list of paths for the extracted FeatureStructures. The feature values found at these paths (or "null", if the path is not valid) are concatenated to generate the name. The special path "—" is used to denote the type of the top-level Feature Structure.

E.g., "— type" concatenates the type (value) of the top-level FeatureStructure and the attribute value of the "type" feature. As with path definitions in `de.lt.tfs.FeatureStructure.getPath`, paths can use as delimiter the "—" or "." (e.g., "FTYPE out.type" would be equivalent to the previous example).

ShUG File A `de.dfki.lt.tfs.ShUG` which tells the Parser relative to which ShUG to construct the FeatureStructures. This will usually be some `*.grm` file.

Ensure XML A `java.lang.Boolean` flag which, iff set to true, tries to filter non-XML contents, as they might occur in SGML corpora (e.g., MUC). The exact semantics of this procedure are defined in the `de.unisb.jtaco.coli.bees`

3.3 Parsing Settings

When using JTaCo with Sprout, you have to select the Sprout components (Tokenizer, Gazetteer, etc.) with which you want Sprout to parse the raw text read from the input corpus. Your Sprout project setup defines the available components. When starting JTaCo, the components displayed for these settings are those available within Sprout at that time. To change them in JTaCo, e.g., when changing to another project in Sprout, you have to make JTaCo `reread` the components from Sprout.

3.4 Comparison Settings

When comparing, JTaCo will usually take into account the tag names, and their character counts (beginning and end) to determine matches between the two annotations. If these three values correspond, a tag from the manual annotation is said to have been correctly identified in the parser output.

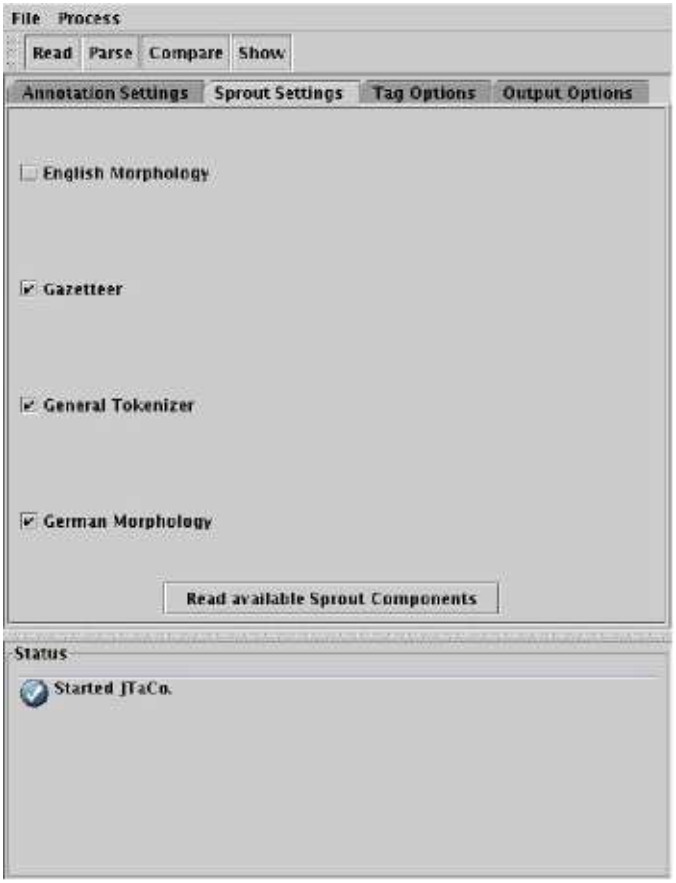

Figure 2: Example components as they would show in the Parser Options panel when using JTaCo with Sprout.

Figure 3: Example comparison settings as displayed by the JTaCoPluggedGui. The first 'Aliases' line defines the manual tag 'A' to be equivalent to a tag 'E' in the parsed annotation. The second line defines a sequence of tags 'D E F' to be the manual equivalent to a sequence of of a 'C D' sequence in the parsed annotation. Here, there can be up to two other tokens between sequence members, and the beginnings of the sequences do not have to be the same to count as a match. In the 'Ignore' table, the tags 'NP' and 'percentage' are excluded from the comparison, although for 'percentage', we still count correctly identified occurrences.

Note, however, that equality can be defined in different way depending on what kind of tags your JTaCo setup uses.

Often, however, the naming scheme of an annotated corpus will differ from the tag names given by the parser. Moreover, sequences of marked entities in the annotation might correspond to single tags in the parser annotation and vice versa. In JTaCo, these correspondences are handled by defining *Aliases* over tags or tag sequences. Such an alias defines which tag or tag sequences in the manual annotation correspond to tags or sequences in the Parser output. Each alias takes one row in the *Aliases* settings table.

manual Tags The tags in the annotated corpus for which this alias is defined. This may be a single tagname or a sequence of names separated by whitespace.

parsed Tags Similar to *manual Tags*, for the annotation produced by the parser.

strictness This setting only applies to sequences (equally to sequences in the manual as to sequences in the parser annotation). It specifies how many single tokens may occurr between the tags of a sequence. E.g., if set to zero, the elements would have to adjoin.

open left This setting can apply to single tags as well as to sequences. It specifies that the beginnings of the two compared elements need not be the same in order to count as correctly identified. This can be used where the manual annotation systematically marks parts which are excluded in the parser output, e.g. if it does not include titles (Mrs., Mr. etc.) in marked names, while the parser does.

open right Similar to the *open left* setting, but for the ends of the given tags.

The *open left* and *open right* settings can be used in conjunction. In that case, the given tags or sequences need only share at least one token in the text in order to count as matching.

Apart from allowing for alias definitions in the described way, it might also be desirable to exclude tags from the comparison altogether. This can be done by listing the tag names in the `ignore` table. A row can contain multiple, white-space separated tag names. The names listed here refer to both annotations. If the `only wrong` box is checked, only false occurrences are omitted, while correct occurrences of the corresponding tags are still counted.

3.5 Output Settings

JTaCo can generate statistical output as well as occurrence lists of correctly matched or misidentified tags. The output settings determine which type of information JTaCo renders for which tags. Statistical information is displayed in one table, occurrence lists may show in many different tables. To make JTaCo group more than one tag in the same row for statistical output or the same occurrences table, you can group them in one line in the output settings, separated by white-space. The key word `ALLTAGS` can be used to group all tags together.

4 Output Table Format

JTaCo renders its output in three different kinds of tables: Statistical information about matches, lists of correctly identified tags and lists of missed or falsely identified tags. Which of these tables JTaCo renders depends on the chosen output settings (see section 3.5).

Note that the tables can be sorted by all their colums by clicking on the respective column header (and Shift-click to sort in descending order).

Statistical Information JTaCo renders statistical information for tags or groups of tags in seven colums: `Tag, prec., recall, f-measure, #manual, #parsed, #man-ok` and `#par-ok`. The first column just contains the tag or tags displayed the respective row. `Tag, prec., recall` and `f-measure` contain precision, recall and f-measure for the tags. The four final columns contain occurrence counts for the `manual` annotation, the `parser` annotation, and the correctly identified occurrences in the annotations, respectively.

Note that because tags in the one annotation can correspond to sequences of tags in the other, the total numbers of correctly identified tags need not correspond for the two annotations over all tags. E.g., if a sequence `A B` is defined as an alias (see section 3.4) for a single tag `C`, a correct identification counts as two matches in the annotation containing the sequence (one for `A` and one for `B`), but only one for the other annotation.

This has implications for the computation of precision and recall: Precision is computed as the quotient of correctly identified tags *with respect to the parser counting* and the total of tag occurrences identified by the parser. Recall, on the other hand, is the number of correctly

Figure 4: Example output options as displayed by the JTaCoPluggedGui. These options would yield statistical information for all tags (i.e., one line per tag). Apart from that, we would get four lists of occurrencess: One containing all correctly identified tags, one with all false tags, and two lists containing only the 'corporation' tag occurrences, the correctly identified ones in the one list, the wrong ones in the other.

identified tags *with respect to the manual counting* divided the total of tag occurrences in the manual annotation.

Occurrence Lists Occurrence lists show information about individual occurrences of tags or tag groups in the two annotations. The tables include the position counts of the occurrences (start and end) as well as the source of the respective occurrence (*manual* or *parsed*).

Note that if a tag was correctly identified as part of a sequence, it will show up multiple times in a correct occurrences list, once for itself, and once for that sequence.

A GUI Error Messages

When JTaCo encounters an error the user can rectify, the user interface will normally display a filled red circle in the status area along with a descriptive message. These are listed below along with possible causes and cures.

Messages displayed with an icon of a filled red circle containing an exclamation mark are exceptional conditions which the user might not be able to correct. Such conditions might have to be inspected by someone familiar with the setup of the different components working together in JTaCo (e.g., the programmer of the parser).

There is no manual annotation to compare.Please read an annotated corpus first.
Occurs when JTaCo is told to compare two taggings and has not yet read any corpus from it could have extracted the manual annotation. You might have to inspect your annotation settings and make sure JTaCo has read a corpus, e.g., by saving the raw text.

There is no raw text to feed to the parser.Please read an annotated corpus first.
Occurs when JTaCo is told to call the parser on a text and has not yet read any corpus from it could have extracted the raw text. Check your annotation settings and make sure JTaCo has read a text from a corpus, e.g., by saving the raw text.

There is no parsed annotation with which to compare.Please call the parser first.
Occurs when JTaCo is told to compare two annotations and has not yet called the parser to generate the parser annotation. Inspect the parser and make sure it returns some result.

No comparison options have been specified.Please set them first.
When using JTaCo with Sprout, comparison options are optional, so

this message might indicate that your JTaCo setup is flawed. When using JTaCo in a different context, inspect your comparison options.

The parsing options have not been set.Please specify them. When using JTaCo with Sprout, this message should not occur; it most likely hints at a flow in your JTaCo setup. Instead, if the parser options are not sufficiently chosen, Sprout will return an empty result.

No valid input corpus available.Please choose the corpus first. Occurs when JTaCo was called to read a corpus from some invalid file or stream. Check if the source is available.

No valid annotation parser options available.Please specify them first. Occurs when JTaCo was called to read a corpus without the minimally needed annotation parser settings. See section 3.1 for details on required options when using JTaCo with Sprout.

No output file chosen.Please choose an output file first. Occurs when JTaCo was told to save the raw text without having been given a file setting of where to store the text. Choose it in the annotation options.

No text to save.Please read an annotated corpus first. Occurs when JTaCo is told to save raw text and has not yet read any corpus from it could have the text. Check your annotation settings and make sure JTaCo has read a text from a corpus.

No results to show. Occurs when JTaCo is told generate output without having been called to compare annotations, or when the output options have not been set to generate any output.