Manual for the Dialogue Annotation

& Research Tool (DART)

Martin Weisser

Version: May 2015

Contents

1 Introduction

The Dialogue Annotation and Research Tool (DART, for short) is a research environment designed to allow the user to annotate and analyse single or multiple dialogues in batch mode, with the ultimate aim to identify speech acts automatically, and thereby create pragmatically annotated corpora that can be investigated from a variety of perspectives. Once a set of dialogues has been analysed, the annotations can be verified and, if necessary, post-edited. In addition to the basic annotation and analysis functionality, DART also provides a convenient way for editing and manipulating many of the linguistic resources required for such analyses in order to improve the annotation results and test relevant hypotheses.

The individual features and options will be discussed and illustrated in the following sections, beginning with a brief overview of the DART environment.

2 Overview of the DART Functionality

We'll begin our brief overview with an 'interface tour'. Figure 1 below provides a quick overview of the DART interface, with a corpus already loaded into the workspace and annotated, where the original files are listed on the left and the output on the right.
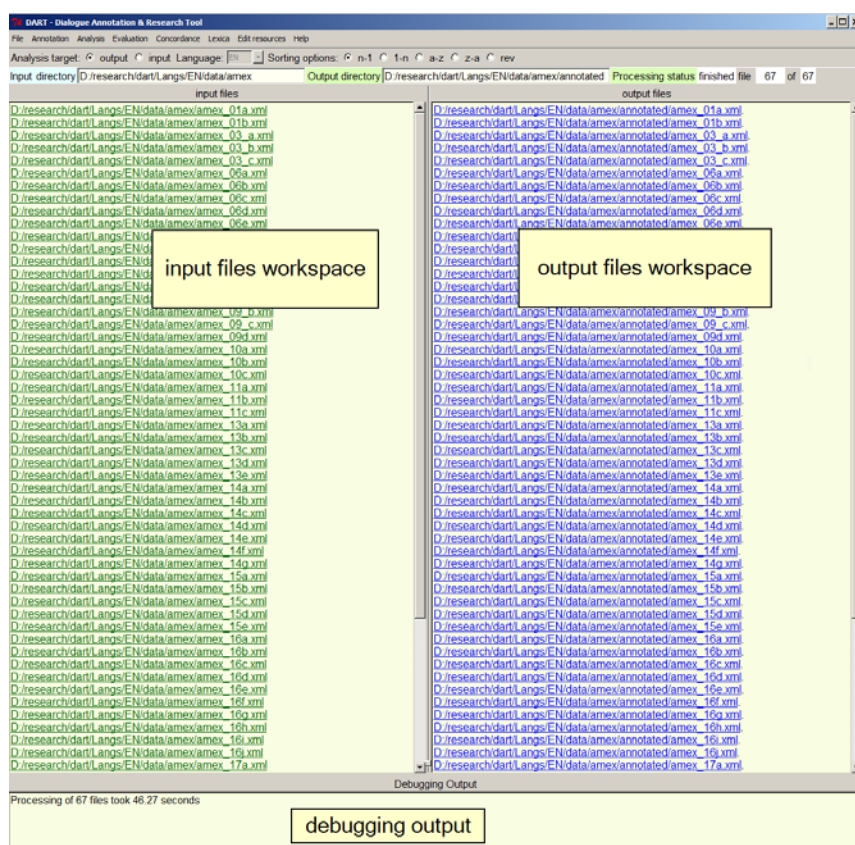
Figure 1 The DART Interface

DART provides facilities for loading a single file or selection of XML files (from a folder or file list; see section 0 below) into the 'input files' workspace, seen in the left-hand window, and running different actions on these. These actions may consist in pre-processing files after loading them in to a built-in editor, which is triggered by clicking on the relevant hyperlink, splitting longer turns, automatically (re-)numbering turns, inserting punctuation tags, etc., or either annotating them pragmatically or tagging them morpho-syntactically in batch-mode. The latter function, however, should not be compared to that of dedicated tag-

gers, such as e.g. CLAWS, as it a) does not come anywhere near the same accuracy as dedicated taggers, and b) uses a tagset that is optimised for use within the pragmatic annotation and other DART-specific routines, such as computer-assisted turn-splitting.

Files loaded either into the 'input files' workspace or the corresponding 'output files' workspace (seen on the right-hand side) can either be analysed using various different analysis routines, such as n-gram generation, concordancing, or extraction of combinations of syntactic tag + speech act frequencies for annotated data, or post-edited in a similar fashion to the pre-processing provided for the input mode, again by opening the relevant file by clicking on the hyperlink and opening it in the built-in editor.

DART also, as much as possible, provides options for editing many of the resources required to perform the analyses/annotation, such as e.g. creating the configuration for a new domain/corpus, editing domain-specific lexica or keyword 'thesauri', syntax definitions, etc. Even though the interface has been developed to be able to control many of these resources easily, even for linguists not versed in programming, some of them currently still require an in-depth knowledge of the system, as well as the Perl programming language it is written in, so that it's probably best to leave those alone until such time that better and more intuitive interfaces can be developed in future.

At the moment, DART only works for English, but the overall design should eventually make it possible to use one-and-the-same system to analyse different languages. The other three languages I've tinkered with so far, and for which language selection options already appear on the toolbar, albeit not available yet, are German, French, and Korean. I hope to be able to implement at least some of the resources for these in the near future.

3   The DART Input Format

The DART input format consists of a very simple form of XML, as depicted in Figure 2 below.
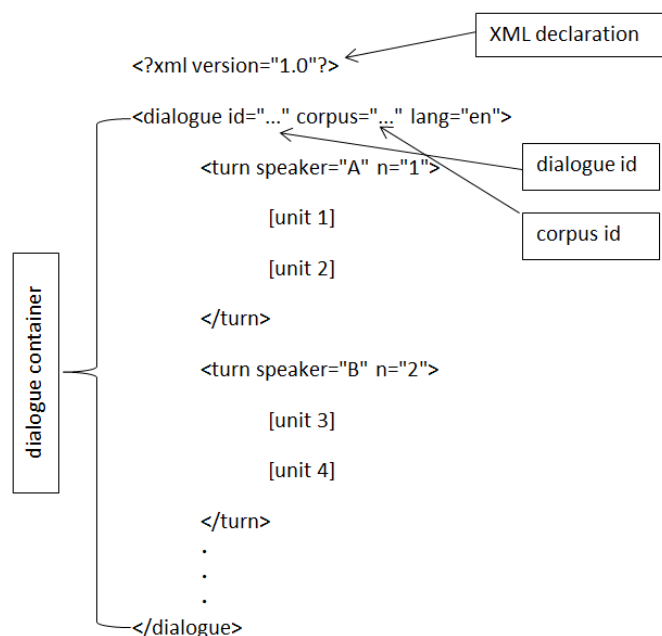
Figure 2 The basic DART XML format

The first part of the file consists of the obligatory XML declaration, followed by the <dialogue>…</dialogue> container element, although an optional style sheet definition may be added in between them. As DART XML files do not have a separate header, relevant meta-information can be stored in the form of attribute–value–pairs in the <dialogue> start element. The ones that are absolutely required for the analysis to run are:

- *id*: a unique identifier for the file within the corpus

- *corpus*: the name of a corpus, which makes it possible to load domain-specific resources, such as e.g. lexica, dynamically

- *lang*: the language of the dialogue, defaulting to *en* for English.

Additional attributes can be freely defined by the user and will simply be copied to the output file, along with the required ones, when the annotated or tagged file is created.

Inside the *dialogue* container, the file needs to be separated into individual <turn>…</turn> containers, each containing two attributes, *n* (for the turn number) and *speaker* (current speaker id). Additional attribute–value pairs may again be added freely by the user. Within the turn, the relevant sentence/clause-like units (C-units) should each appear on a separate line, i.e. separated by a newline from bone another and the XML tags. XML purists may frown upon this, but this simplifies the processing in DART, and as XML is only used as a widely accepted output format, and any XML parser or rendering browser can easily ignore the extra whitespace, this design option was chosen for the sake of expedience and to make the output more easily readable for pre- or post-processing without the need for any additional rendering engine.

All words, apart from proper names should be lowercased in order to simplify the processing.

After pragmatic annotation, each line representing a relevant syntactic and pragmatic unit will end up being wrapped in another XML tag whose label depends on the syntactic category identified by DART, as well as a number of attributes. The syntactic categories currently defined are:

- *decl*: declaratives

- *q-yn*: yes/no questions

- *q-wh*: wh-questions

- *imp*: imperatives

- *address*: terms of address

- *dm*: discourse markers (henceforth DMs)

- *exclam*: exclamations

- *yes*: yes responses

- *no*: no responses

- *frag*: fragments, i.e. syntactically ungrammatical or incomplete/elliptical syntactic units

The attributes for each syntactic element, where not all need to be present in the output, are:

- n: the number of the relevant unit, numbered consecutively and independently of turns

- sp-act: the speech act or speech-act combination identified by DART

- polarity: surface polarity

- topic: semantic information pertaining to what is being talked about

- mode: semantico-pragmatic (interpersonal) information indicating the form of interaction

DART was originally designed for the analysis of unpunctuated text, but of course using sensible punctuation markers for major syntactic or prosodic units helps to disambiguate the options for speech acts, so empty XML elements indicating punctuation (e.g. <punct type='stop' />) can now be included at the end of a C-unit to facilitate the analysis. The *type* attribute value options recognised by DART routines within <punct /> elements are:

- stop: indicates finality, generally in declaratives

- query: indicates interrogative nature, usually occurring with either syntactically marked questions or prosodically marked declarative questions

- level: indicates non-finality, possibly in lists

- incomplete: indicates that the speaker has not completed the unit

- exclam: indicates exclamations or imperatives

Other empty XML elements that are currently defined are:

- \<pause /\>: attribute *length* or none, if length unknown

- \<backchannel\>: attribute *content*

- \<overlap\>: attributes *n* and *type* (options: *start* & *end*)

- \<unclear /\>: attribute *length* or none for single words

- \<comment /\>: attributes *type* (e.g. *phon*(etic) or *syntax*) and *content*; also *sounds_like* for phonetic transcriptions

- \<vocal /\>: for vocal 'noises', such as coughs, etc.

- \<event /\>: any non-linguistic event, such as background noises

Additional categories can be freely added, either by typing them in or by adding additional buttons or values to the pre- or post-processing options in the built-in editor (see section 0 for details). All empty elements, apart from \<pause /\> and \<punc /\> are simply ignored in the processing and written to the output file unchanged.

4   Processing Options for Annotation & Tagging

4.1   Adding a New File

A (new) sample file, including two pre-defined turns, can be created through the 'File→New' menu option.
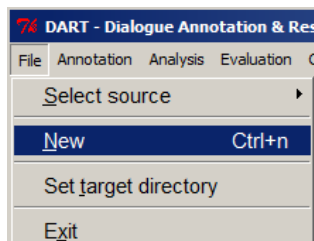
Figure 3 Dialog for creating a new, 'blank', annotation file, containing two turns

As in most windows programs with editing functionality, the keyboard shortcut for this is 'Ctrl+n'.

4.2   Selecting File(s) for Processing

Corpus data in the form of single or multiple files for pre-processing, annotation, or analysis can be loaded via the 'File→Select Source' menu.
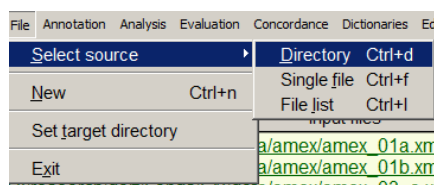
Figure 4 Source selection options

As Figure 4 above shows, selecting the 'File→Select source→Directory' option or pressing the keyboard shortcut 'Ctrl+d' allows for a whole directory to be selected (see Figure 5), while choosing 'Single file' (Ctrl+f) or 'File list' (Ctrl+l) will open a file selection dialogue (see Figure 6).
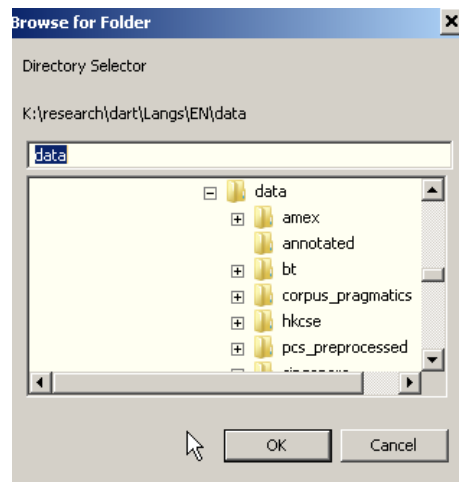
Figure 5 Directory selector

If no directory is chosen, and the dialogue cancelled, or the user presses the 'Esc' key, by default, files in the 'test' directory will be selected.
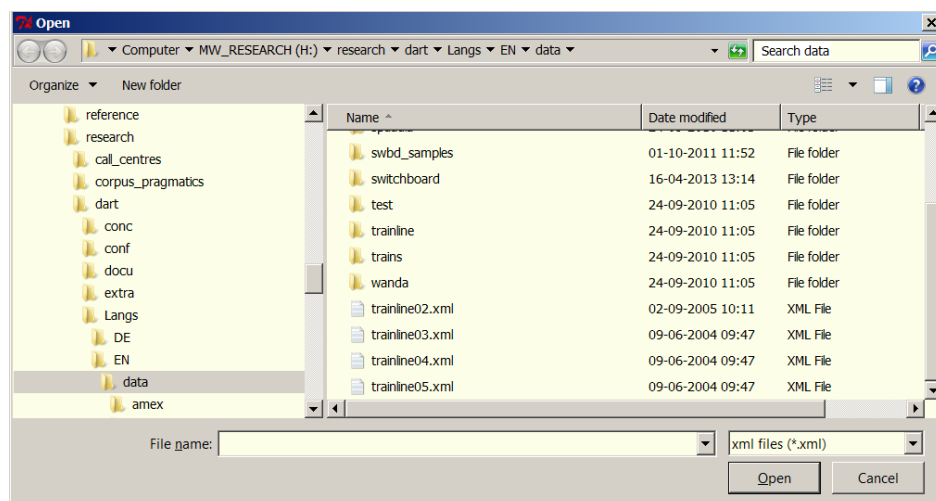
Figure 6 File selector

Cancelling this dialogue will simply return the user to the main DART interface. The dialogue for selecting a list of files essentially looks the same as in the above figure, only showing files with the extension '.lst'. Once a file or directory has been selected, the relevant file(s) are added to the input files workspace as hyperlinks that open the relevant file in the built-in editor, as well as added to a list of files to be processed in the system. The source directory is also indicated in the source directory box above the workspaces, and the default

target directory set to a sub-directory of the same directory named 'annotated' (or 'tagged' in tagging mode), as depicted in Figure 7.

Figure 7 Source and target directory indicator

If the output target directory doesn't exist, it will be created once an annotation or tagging process is started. The processing status indicator towards the top right-hand corner will also reflect the number of files selected, as illustrated in Figure 8.
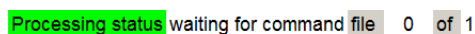
Figure 8 Processing status indicator (prior to processing)

This processing status indicator will change to reflect the number of files processed once a processing operation has been triggered.

4.3   Pre-Editing Input Files

Any file that appears in the input files workspace can be opened in the built-in DART editor by clicking one of the hyperlinks in the left hand window. Before any annotation has been carried out, this will usually be done to pre-edit the file to break long turns or to fix issues that may have arisen from an automatic conversion process to the DART format.
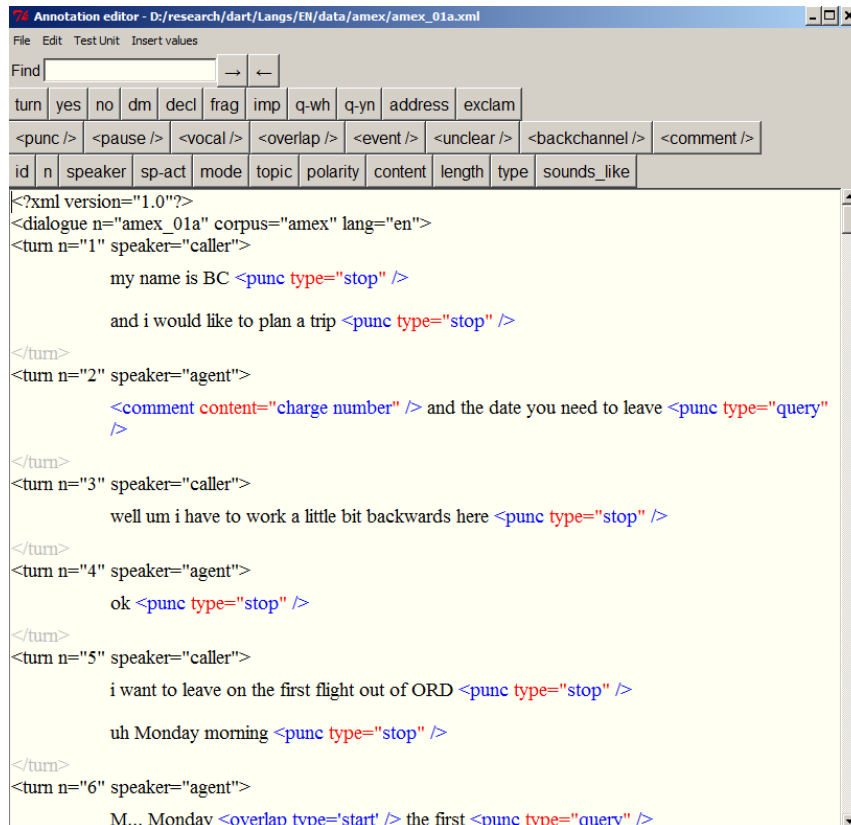
Figure 9 The built-in editor showing a file opened for pre-editing

The editor has standard key bindings for saving ('Ctrl+s'), copying ('Ctrl+c'), cutting ('Ctrl+x'), pasting ('Ctrl+v'), as well as undo ('Ctrl+z') and redo ('Ctrl+y') operations. To close the window, the binding 'Ctrl+w' can be used, and 'Ctrl+S' is a shortcut for saving and closing the window in one go, while 'Ctrl+f' puts the keyboard focus in the find box for the user to type in a search string. The forward search can be triggered via the Enter key, as well as by clicking the forward arrow (→) next to the find box, while the box has the focus, whereas a backward search can only be triggered by clicking the backward button (←).

For ease of recognition, and to be able to distinguish the main text from them, empty elements are colour-coded in blue, while their attribute names are highlighted in red.

At this particular stage of dealing with corpus data, perhaps the most important automated functions in DART are provided through the 'Edit' menu, shown below.
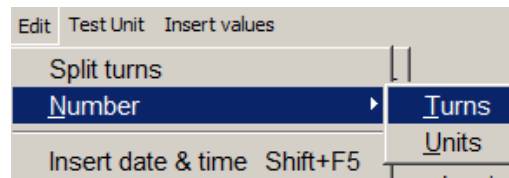
Figure 10 The 'Edit' menu of the built-in editor

As often longer turns in fact do constitute multiple pragmatic units, DART provides facilities for doing two different things, a) a fully automated splitting off of turn-initial DMs or yes/no-responses during the annotation process itself without any manual assistance, and b) a computer-assisted turn splitting process that suggests potential splitting points to the user. This second option is triggered from within the editor window via 'Edit→Split turns', and the corresponding dialogue depicted in Figure 11.
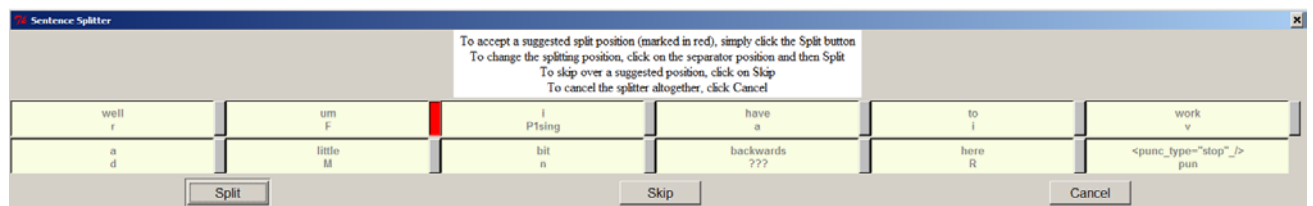
Figure 11 The sentence splitter dialogue

As the above figure shows, the user can either accept a suggested splitting position, skip over it, or change it by clicking on one of the separator position buttons, while clicking on 'Cancel' will exit the splitting routine altogether.

Another useful option is hidden behind the 'Number' menu, with its associated sub-menus 'Turns' and 'Units'. In a pre-editing context, the 'Turns' sub-menu is the most relevant, as it may often be necessary to split turns where an automated conversion program has incorrectly assigned units within the same turn to one speaker, or merge them, e.g. where backchannels have erroneously been handled as separate turns, thereby artificially breaking a speaker's turn. As it would be very cumbersome to manually re-number all the turns in such cases, especially in a rather lengthy dialogue, the editor will take care of this re-numbering

automatically via this menu option, returning the user to the last edit position once the process has been completed. Re-numbering units is only relevant after the pragmatic annotation process has been carried out.

The final menu option on the 'Edit' menu, 'Insert date & time' can be used for time-stamping edits, either if an appropriate attribute has been added to the dialogue container, or for adding the relevant information to comment lines in one of the resource files (see section 0).

For manual editing inside the editor, there is a set of user-configurable/-editable toolbars (see Figure 12) and (context-)menu items (see Figure 13).

turn	yes	no	dm	decl	frag	imp	q-wh	q-yn	address	exclam		
<punc />		<pause />		<vocal />		<overlap />		<event />		<unclear />	<backchannel />	<comment />	<tone />
id	n	speaker	sp-act	mode	topic	polarity	content	length	type	sounds_like	content, <type> (phonetic, syntax, restart), <sounds_like>
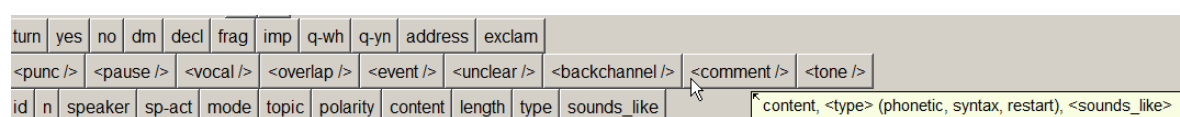
Figure 12 The toolbars in the built-in editor

The top toolbar contains a set of dialogue-relevant non-empty XML elements that can either just be inserted or wrapped around selected textual content. Most of these, though, apart from <turn>, relate to dialogue elements that will only be relevant for post-editing, in order to revise parts of the automated annotation. The second row contains a set of empty XML elements that may be used predominantly in pre-editing mode, usually in combination with one or more of the attributes inserted by clicking buttons found in the third row. As can be seen in Figure 12, the configuration files are designed to contain information that can be rendered in the form of a tooltip for each button to indicate the required and optional attributes, as well as their potential values.

To prevent the user from having to type in potential attribute values, thereby potentially also producing typos, as well as to some extent constraining pre- or post-editors to only use a set of fixed options to ensure consistency across a project, attribute values can be inserted

from both the 'Insert values' menu and also via the context (right-click) menu. The options

for the former are shown in Figure 13.
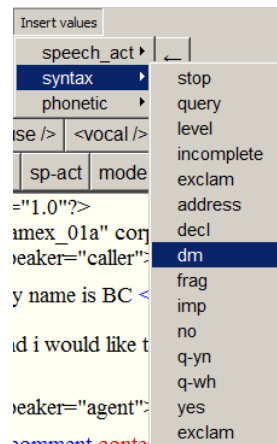
Figure 13 The 'Values' menu in the annotation editor

The two sets of these values that are essentially relevant in a pre-editing context are 'syntax'

(see Figure 13) and 'phonetic', where the former can be used to add values to punctuation

elements, and the latter to transcribe phonetic phenomena when using the 'sounds_like' at-

tribute.

Figure 14 The 'phonetic' context menu

The remaining option, 'speech act' only becomes relevant in a post-editing context, once a pragmatic annotation has been carried out.

The 'Test Unit' option, depicted in Figure 15, allows the user to get a 'sneek preview' of how the unit would be annotated.

Figure 15 The unit tester

Although this feature is mainly intended to make it possible to verify why a particular unit has not been annotated with an expected speech act after the pragmatic annotation process has been carried out, it can also be used to identify gaps in the lexicon, weaknesses in the tagset or syntax rules, as well as to test whether the text-cleanup routines successfully remove all unwanted material, such as backchannels or comments from the unit to be analysed. As the analysis is only conducted for a single unit, however, the inferencing process that is normally conducted to identify the final speech-act label cannot be carried out fully to determine contextual features, such as identifying whether a unit functions as a response, etc., and thus such 'secondary' speech acts are not listed.

4.4 Annotating or Tagging Files

DART provides two options for annotating files loaded into the input files workspace. The main one is the one labelled 'Pragmatic', which is triggered by selecting the relevant option from the 'Annotation' menu or pressing 'Ctrl+a'.
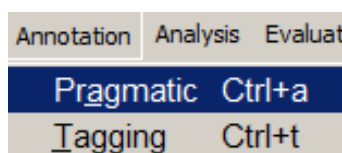
Figure 16 The annotation options menu

The second option, 'Tagging' (shortcut 'Ctrl+t'), can be used for both un-annotated and annotated files. This tagging, however, as pointed out above, is fairly rudimentary, and its accuracy can and should in no way be compared to that of a dedicated tagger such as CLAWS. Its main function is to verify issues pertaining to the pragmatic annotation process, such as finding incorrectly identified syntactic units, due to errors in the syntax rules or out-of-lexicon words.

All files selected for processing are annotated in batch mode without any further user input, unless of course an error occurs in the processing, in which case one or more error messages will be written to the 'Debugging Output' window at the bottom of the DART main window, and the annotation process may be interrupted altogether, depending on the seriousness of the error, or whether some form of error handling has successfully been implemented. Most frequently, such errors relate to ill-formed XML, or incorrectly/incomepletely specified resources, such as e.g. a *lang* or *corpus* attribute missing from the 'header' of a file. To avoid any well-formedness related errors, the user should ideally check corpus data loaded into the workspace before running any annotations by selecting 'Ev<u>a</u>luation→Check <u>w</u>ellformedness'. Missing or incorrectly specified *lang* or *corpus* attributes will be reported during the annotation process itself, though.

Having the appropriate attributes for *lang* and *corpus* present in the container element is essential because, only in this way, all the necessary resource files can be dynamically loaded while reading in dialogue files for annotation.

During the annotation process, the processing status indicator is constantly updated upon completion of the annotation of each file, and a hyperlink to the resulting file is inserted in the output files workspace. Even annotating relatively long files of more than 100 turns should generally take no longer than about a maximum of 20-30 seconds, so if the processsing indicator has not changed for a longer period of time, it is likely that an error may have occurred, in which case the user should first check the debugging output for errors to fix, and, in the worst case, close and restart DART after this.

4.5 Post-editing Annotated Data

Just as for pre-editing, files for post-editing are opened in the built-in editor window by clicking one of the hyperlinks, this time in the right-hand 'output files' workspace window,

at least if they're opened immediately after the annotation process has been completed. For post-editing already annotated files, these can be opened in the input files workspace.

A pragmatically annotated file will look different from an un-annotated one because additional XML elements will have been wrapped around the individual syntactic units and also because these elements are colour-coded in order to reflect at least part of their (semantico-)pragmatic potential, which can be seen in Figure 17.
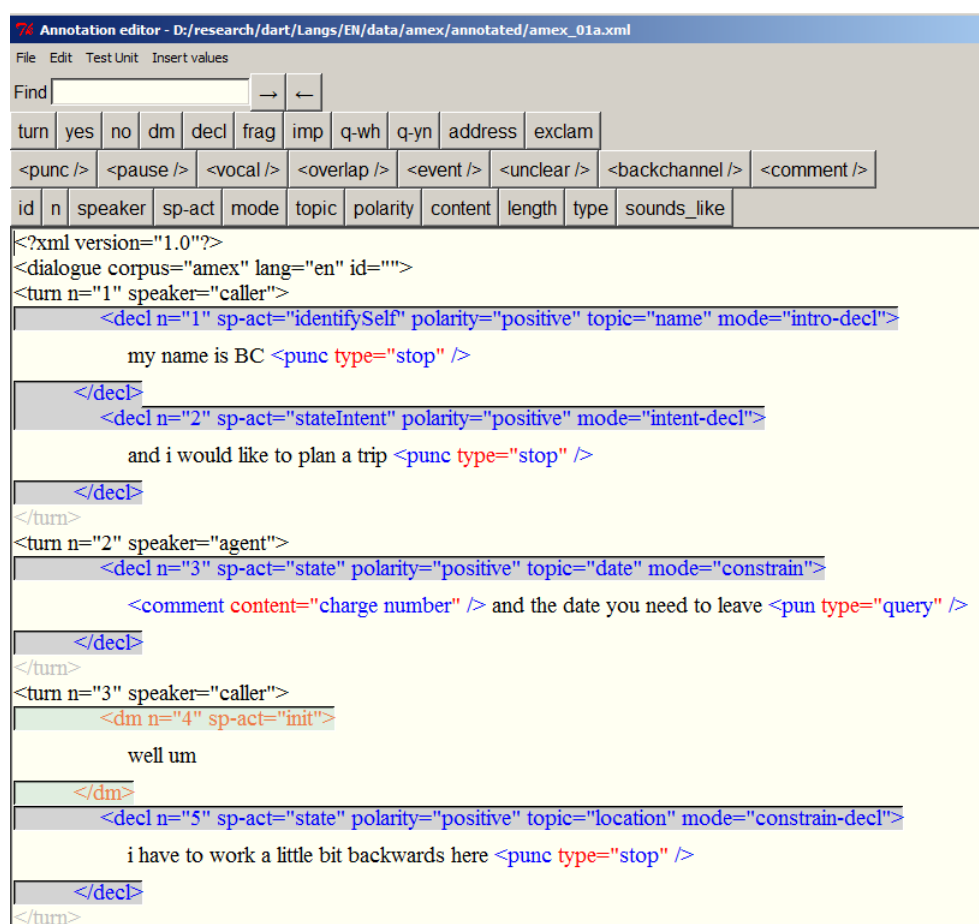

Figure 17 The built-in editor showing a file opened for post-editing

The colour coding semantics are as follows:

- *decl*: grey background, blue text

- *q-yn*: ivory background, red text (signifying closed set of options for answers)

- *q-wh*: ivory background, green text (signifying open set of options for answers)

- *imp*: light blue background, red text (signifying imperative force)

- *address*: orange background, dark blue text

- *dm*: greyish blue background, orange text

- *exclam*: light-blue background, orange text

- *yes*: blue background, white text (mnemonic symbolising a 'forward-moving' option, as in an 'entry permitted' traffic sign)

- *no*: red background, white text (mnemonic symbolising a blocking option, as in an 'no entry' traffic sign)

- *frag*: light blue background, black text

Beginnings of turns appear in black text colour, but their ends are 'greyed out', so as to make them less prominent. Any textual material appears in black text colour, without any special background.

4.5.1 General Editing Options

In general, the post-editing will consist in fixing any potential errors the program may have made in the annotation phase. Most of these errors will generally pertain to incorrectly or incompletely identified speech acts, but some of them may also be related to other levels of annotation, such as e.g. errors in the syntactic annotation, in case the built-in rule set has not been able to identify the nature of a unit, e.g. 'mis-interpreting' a declarative as a fragment, etc.

In case a syntactic element has been misidentified, the corresponding tag name can simply be retyped or selected from the 'syntax' context menu (see Figure 18), shown below, as well as from the corresponding item on the 'Insert values' menu:
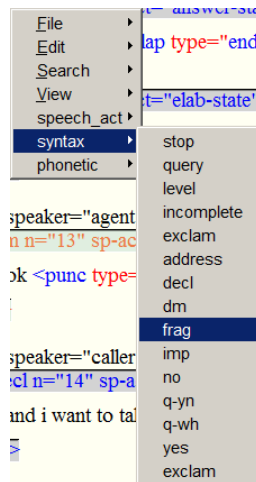
Figure 18 The 'syntax' context menu

To add or replace speech-act values, items can be selected from the relevant context menu item (see Figure Figure 19 below), too.
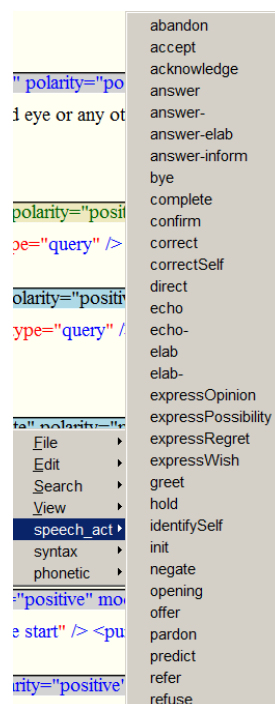
Figure 19 The speech acts context menu

If, for some reason, post-editors may decide that a unit that was either not manually split in the pre-editing or by some of the automatic splitting routines of the annotation process, then they can add a manual line break between the units and move any relevant element items to

be retained via copy-and-paste, and then add a new syntactic element plus attributes by using the toolbar options for wrapping text in element tags and creating attributes.

In some cases, for instance if turns have not been correctly marked in an automatic conversion of dialogue data to the DART format, the user can also manually break turns, and then insert the appropriate turn tags and attributes, although, ideally, such issues should be identified and fixed in the pre-processing stage of a corpus.

4.5.2 (Re-)Numbering Turns or C-Units

In both of the last two cases described above, for turns or units, respectively, the numbering would of course then be off and need to be adjusted. As this would again be extremely tedious and error prone if done manually, the editor provides functions for updating the numbers automatically from the 'Number' item on the 'Edit' menu. These functions can be triggered from anywhere within the text, and will automatically position the cursor again at the relevant point after the update operation has been completed, so that post-editing can continue smoothly.

5 Frequency Analysis Options for Annotated or Un-Annotated Data

The 'Analysis' menu options currently provide options for generating basic descriptive statistics on either annotated or un-annotated corpora.
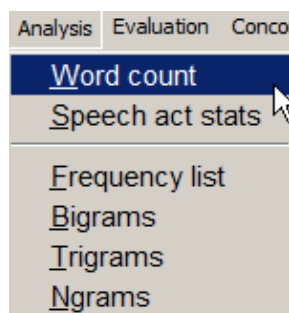
Figure 20 The 'Analysis' options menu

The most basic statistic here is a simple *word count* of the basic textual content, i.e. excluding the contents of tags or empty elements. The remaining items are all related to functions that produce n-gram statistics that may either aid in the analysis of dialogues for research purposes or in creating/testing resources that are used in the annotation process, as well as verifying annotation results.

5.1 Basic Frequency Analysis (N-grams)

Ngram analysis is an important means for identifying recurring lexico-grammatical patterns on different linguistic levels in dialogue corpora, which, once they have been discovered, can then be modelled and integrated into the analysis. The 'Analysis' functions in DART provide access to this for creating frequency/n-gram lists for a given corpus.

Options for creating a basic (unigram) 'Frequency list', 'Bigrams' and 'Trigrams' are already predefined, while selecting 'Ngrams' allows the user to specify how long the sequences of words to be counted should be. Especially the latter option makes it possible to inves-

tigate longer *phraseological patterns* or units that may be relevant on the levels of semantico-pragmatics or semantics. Once such recurrent 'themes' have been identified roughly, their general patterns of occurrence, along with any inherent variability, can be deduced and regular expressions for pattern matching created and grouped under suitable category labels. If such a pattern can be matched against a textual unit during the annotation process, the corresponding label can be recorded and later potentially be used as part of the inferencing process in identifying speech acts. If, on the other hand, the information matched in this way is purely semantic and therefore does not contribute to the interactional strategies of the speakers, it can still be recorded as a conversational topic and may later help to identify particular stages of a dialogue.

To make it easier for the user to verify any hypotheses developed from looking at the frequency data, all entries are hyperlinked, so that they trigger the built-in concordancer (see 0 below) with pre-defined search terms. Figure 21 below illustrates the basic output format in the form of a trigram list.

Figure 21 A trigram frequency analysis

For creating the basic frequency lists, a number of different sorting options exist to produce different 'views' of the data.
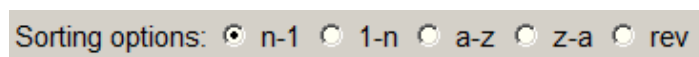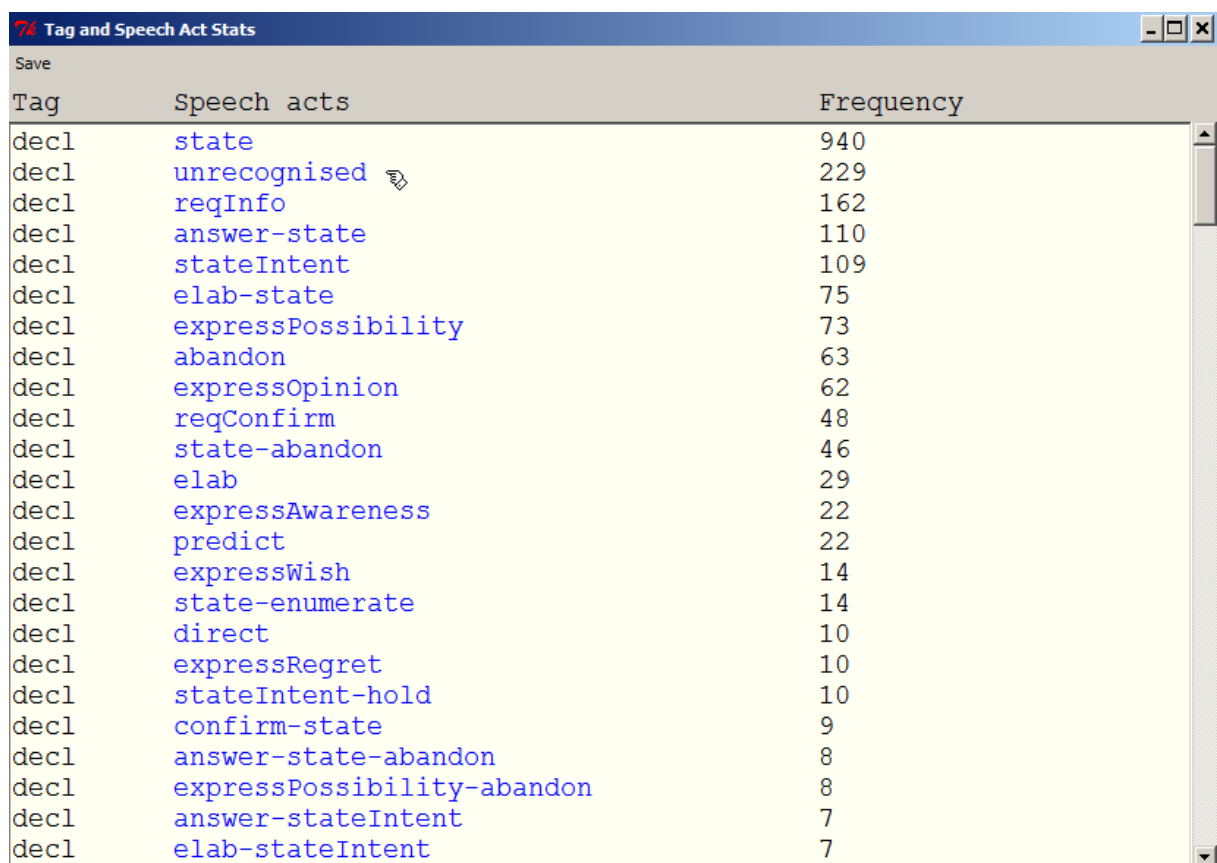
Figure 22 Sorting options for frequency lists

The default option is a frequency list sorted in descending order of frequency (*n-1*), with a secondary *asciibetical* sort order (i.e. words with initial capitals will be sorted before words with non-capital initials). This can be reversed (*1-n*) to list low-frequency items, such as *hapax legomena*, first, perhaps in order to identify highly domain-specific content. The next two options are for asciibetical sorting (*a-z*) and its counterpart (*z-a*). While the first four options, as already pointed out above, are mainly useful for identifying vocabulary or phraseological patterns, the final sorting option (*rev*) creates a reverse-sorted list which may be

used for identifying morphological features that can be modelled in the morpho-syntactic analysis (tagging) routines.

5.2 Speech-Act Frequency Analysis

The first entry on the 'Analysis' menu, 'Speech act stats', of course only makes sense to use if a corpus has already been annotated. The output from this lists speech acts according to the syntactic tag they occur with, alphabetically sorted first according to the tag, then sorted in descending frequency of the speech act itself, plus the raw frequency of occurrence in the corpus. Some sample output is shown below:

```
Tag and Speech Act Stats                                      _ □ ×
Save

Tag          Speech acts                        Frequency
decl         state                              940
decl         unrecognised                       229
decl         reqInfo                            162
decl         answer-state                       110
decl         stateIntent                        109
decl         elab-state                         75
decl         expressPossibility                 73
decl         abandon                            63
decl         expressOpinion                     62
decl         reqConfirm                         48
decl         state-abandon                      46
decl         elab                               29
decl         expressAwareness                   22
decl         predict                            22
decl         expressWish                        14
decl         state-enumerate                    14
decl         direct                             10
decl         expressRegret                      10
decl         stateIntent-hold                   10
decl         confirm-state                      9
decl         answer-state-abandon               8
decl         expressPossibility-abandon         8
decl         answer-stateIntent                 7
decl         elab-stateIntent                   7
```

Figure 23 Sample speech-act frequency output

As can be seen from the sample above, the frequency list also includes information as to how many and which syntactic units were left unlabelled (marked *unrecognised*) on the speech-act level. The frequency list in general allows the researcher to gain an overview of

the distribution of particular dialogue features and to identifying potential analysis problems or gaps in the system's rules for a given corpus/domain, especially in cases where it has not been possible to assign a relevant speech act.[1]

---

[1] The relatively high number of unrecognised units is due to the fact that the corpus I used for illustrative purposes here was not pre-processed. Nevertheless, DART managed to classify 90.37% out of a total of 8,595 units, leaving only 9.63% marked as unrecognised.

6 Concordancing & Editing

DART also incorporates an integrated (line-based) concordancing facility. This can be used to investigate particular expectations or hypotheses concerning the nature of the data the user may have for unprocessed data, or explore annotated or tagged data in order to verify or test the effectiveness of the different processing routines at different levels. The concordance module takes the currently 'activated' set of files as its input, i.e. if an annotated or un-annotated corpus has been loaded into the input file list, this will be used, while this is switched to the output file list if an annotation or tagging process has already been carried out. It is, however, possible to switch this manually, too.

Figure 24 shows an illustration of a concordance output for the combined search for the <dm> tag in combination with the target word *init* that finds all occurrences of discourse markers that 'initiate/initialise' a new topic.
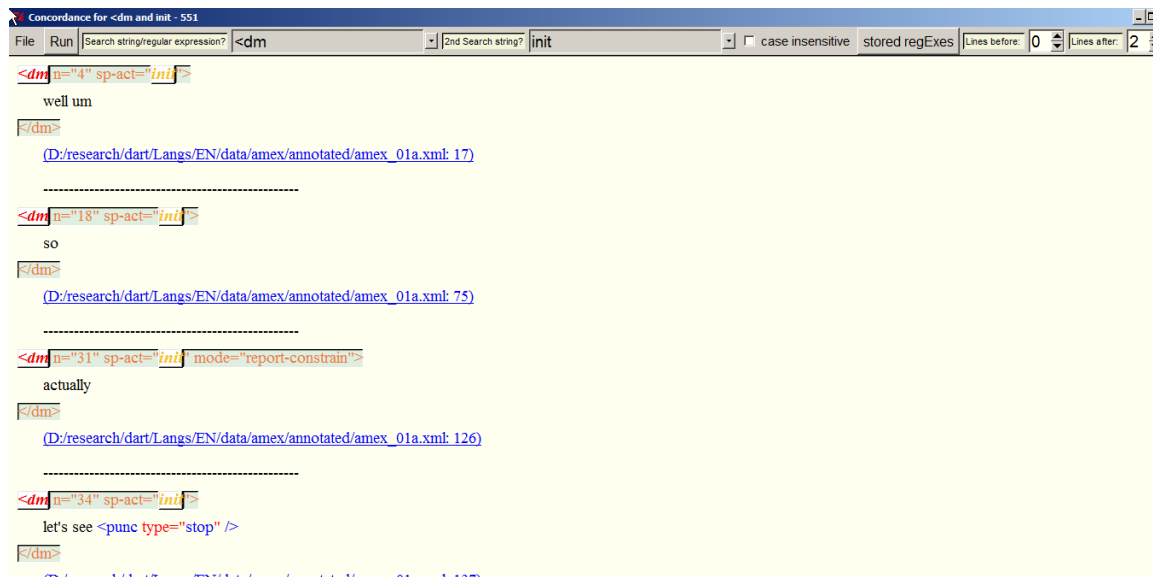

Figure 24. Concordance output for <dm and search term 'init'

As the illustration above shows, the concordancer currently provides the option to look for a combination of two *search terms* (patterns), although providing only a single pattern is of course possible, too. While terms can be specified freely, some of the most common terms

of interest for the analysis of annotated dialogues are already predefined via drop-down lists, where the first list contains the beginnings of all syntactic elements and the second a list of potential attributes. A set of commonly used example search terms can also be saved to a file that can be accessed and edited via the 'stored regExes' button. If terms are specified freely, the full, rich set of *Perl* regular expressions can be used, which makes it possible for the advanced user to specify highly complex patterns to search for, but at the same time still enables the 'average user' to simply 'look for words'. Searches can be made case insensitive, which may be required for dialogue data that has been transcribed using uppercase characters for stressed syllables.

The output consists of the relevant target line with the search term(s) highlighted, and 'surrounded' by a specified variable context of *n* numbers of lines preceding or following the target line itself. The context is controlled via the spinners next to the 'Lines before' and 'Lines after' labels. This make it possible to easily widen or collapse the context in an appropriate manner, for instance if one wants to investigate different types of responses and their 'triggers' in the previous speaker's turn in an analysis of *adjacency pairs* or *IR(F)* (initiation-response-feedback) sequences. Each hit is also followed by an indication of the file name and line number where it was found in round brackets. An indication of the number of hits is always provided, both in the title bar of the concordancing window and as a summary at the top of the text file output.

The information from the concordancing window can be copied and pasted straight into a word-processor if the user wants to e.g. extract only a few examples from a corpus for use in a publication or, alternatively, a complete list of all concordance hits saved to a text file. However, what is perhaps even more useful in the annotation context is that the source information for each individual hit again triggers a hyperlink that opens an editor window at the relevant line number, and with search term highlighted, allowing efficient on-the-spot

post-editing. This can, for instance, be used to automatically identify and select all occurrences where DART has been unable to assign a speech act to a unit, and thus quickly fill the annotation gaps. Another use for this would be if a user who, due to a lack of programming knowledge, does not have the expertise to fix particular analysis rules that have been identified as misfiring, but has the linguistic understanding to identify these cases in the results, could select all miscategorised instances and correct them in this way.

7   Evaluation Options: Sample Selection & Consistency Checking

When having to perform some form of evaluation on the data that have been analysed, researchers are often faced with the issue of having to select an unbiased sample. DART facilitates this task by allowing the user to specify a fixed number of files to be selected randomly from a given data set loaded into the input files workspace. These files can then be investigated in various ways, either 'manually' or using the DART analysis features described above. This feature is triggered by selecting 'Choose files' from the 'Evaluation' menu depicted below.
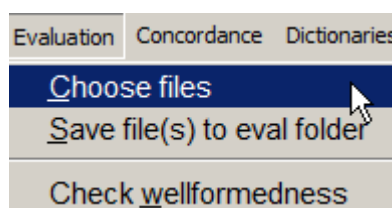
Figure 25 The 'Evaluation' menu

Upon clicking this option, the user is prompted to input the number of files to be chosen randomly, and the list of files will then be written to the output files workspace. The next option on the menu, 'Save file(s) to eval folder', will then copy the selected files to a special folder called 'eval' within the 'data' folder for the relevant corpus, also clearing the output files workspace. In the course of this process, a test is also run on the 'eval' folder to determine whether it already contains any files, in which case the user can either specify to delete any existing files, add the newly chosen set, or cancel the operation, in case they want to manually back up any existing evaluation files, e.g. to a sub-folder within the 'eval' folder, because DART only deletes files in the main folder, but not (sub-)folders.

The final option, 'Check wellformedness', allows the user to test the well-formedness and consistency of the XML annotations to identify errors related to the XML structure of the corpus files. Such errors would otherwise prevent the files from being displayed in a web browser or processed by other applications that 'understand' XML coding if the corpus data is passed on to other researchers who do not have access to DART. Any error messages resulting from this check are written to the Debugging Output window at the bottom of the screen.

The debugging output window can also be used by the expert user to output interim analysis results or other information for debugging purposes. Any error messages produced by the operating system or Perl (which DART is written in) are also redirected there to help the expert user to fix programming errors, and would allow the naïve user to copy and paste them to pass them on to a more experienced one.

Furthermore, at the end of each annotation or tagging operation, an indication of the length of the overall process will be provided in order to be able to evaluate the efficiency of the tool in a rather rough manner. It is important to note, though, that this output only provides a relative measure of efficiency and does not represent a true benchmark, as various factors may influence the behaviour of the tool. Amongst those are the speed of the processor, to some degree the caching mechanisms of the perl interpreter, the amount of debugging information output generated, etc.

8 Creating & Editing Resources

Being a research environment that tries to incorporate as many of the tasks involved in analysing and annotating dialogues, DART also provides convenient interfaces and mechanisms that enable the researcher to create and update the relevant resources as much as possible. The following sections will describe the different levels on which this is currently feasible – or advisable – in some more detail.

8.1 Changing Configuration Options

One of the central notions in the design of DART is that of *genericity*. This means that many of the resources that are employed in analysing dialogues are initially reduced to a *common core* of elements that represent recurrent features of dialogues on different levels. These resources are always available and already make it possible to analyse dialogues from new domains to a relatively satisfactory extent without creating or using additional resources. Augmenting the generic core by suitable, more domain-specific resources then generally increases the quality of the analyses further. Some of the relevant areas where this notion is applicable will be discussed further below.

As DART has thus been designed to maximise the flexibility in adapting its analysis resources to different domains, as well as (at least theoretically) different languages, although this has not been fully implemented yet, there are a variety of features that make it possible to create new resources for this purpose, and to later 'pull in' these resources at runtime. The 'Edit resources' menu allow the user to create new domain/corpus folders, and set or edit different domain-specific properties, for instance the topic files or domain-specific lexica, as can be seen in the illustration below.
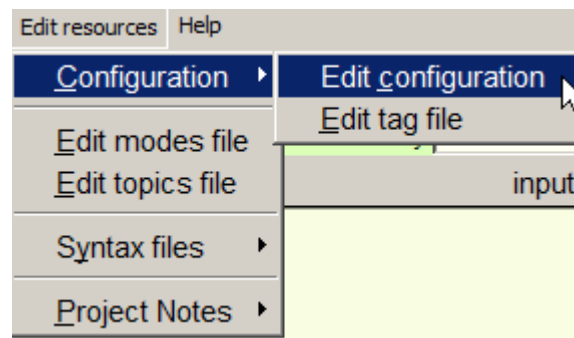
Figure 26 The 'Edit'resources' menu

Clicking the 'Edit configuration' menu item will open up the 'Configuration editor' window, as illustrated in Figure 27.
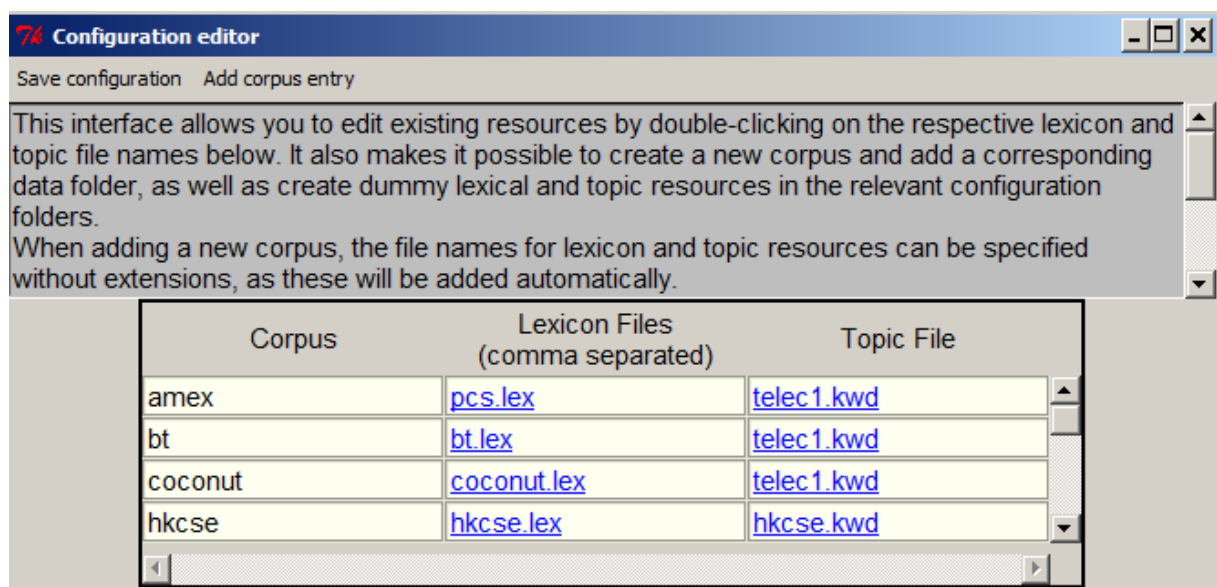
Figure 27 The Configuration editor

Through this interface, the user can add a new domain/corpus by selecting the 'Add corpus entry' option. This will add a new row to the configuration table, where the name of a corpus, as well as (optionally) the names for one or more lexica (comma-separated), as well as a topic file, can be specified. Once the user chooses the 'Save configuration' option, a new folder with the corpus name will be created (if it doesn't already exist), along with (currently) 3 sub-folders named 'info', 'notes', and 'stats'.

The first of these can be used to store meta information on the corpus and the speakers. Currently, none of DART's features do make use of this, but future versions will probably include facilities for accessing speaker properties, creating group definitions for speakers (e.g. call-centre agents & customers, teachers & students, etc.) to allow the user to create comparative descriptive statistics for individual speakers or speaker groups, or potentially define various levels of 'authority' for a given speaker to indicate whether there's a hierarchical structure among/between speakers, which may, for instance, allow DART to 're-define' a speech act *directive* as a command, etc.

The second folder, 'notes', can be used to collect and store textual notes regarding different observations related to the data, while the 'stats' folder can be used to store the results of any frequency analyses conducted on the corpus.

If filenames have been specified for lexica and topic files, 'dummy' files, containing only information about the entry format and date & time of creation, are created in the 'Lex' and 'Topics' folders for the relevant language[2], respectively, if no file of the same name exists. However, for lexicon files, usually a better option is to 'synthesise' a new lexicon from a corpus loaded into the input files workspace (see 9.2 below). All lexicon and topic files that already exist when the interface is opened are also hyperlinked to opening the relevant file in the editor.

The 'Edit resources' menu also provides access to an editing function for specifying/editing the different types of XML elements and attributes, and PoS tags that appear as buttons or menu items inside the editor windows for pre- or post-processing XML or lexicon data. This is done through clicking on the 'Edit tag file' menu entry and selecting the relevant file for elements, attributes, values, or PoS tags, which all have the extension '.cnf' (i.e.

---

[2] As stated before, to date only English is supported.

configuration). A sample of the editing options for these resources is depicted in Figure 28 below.
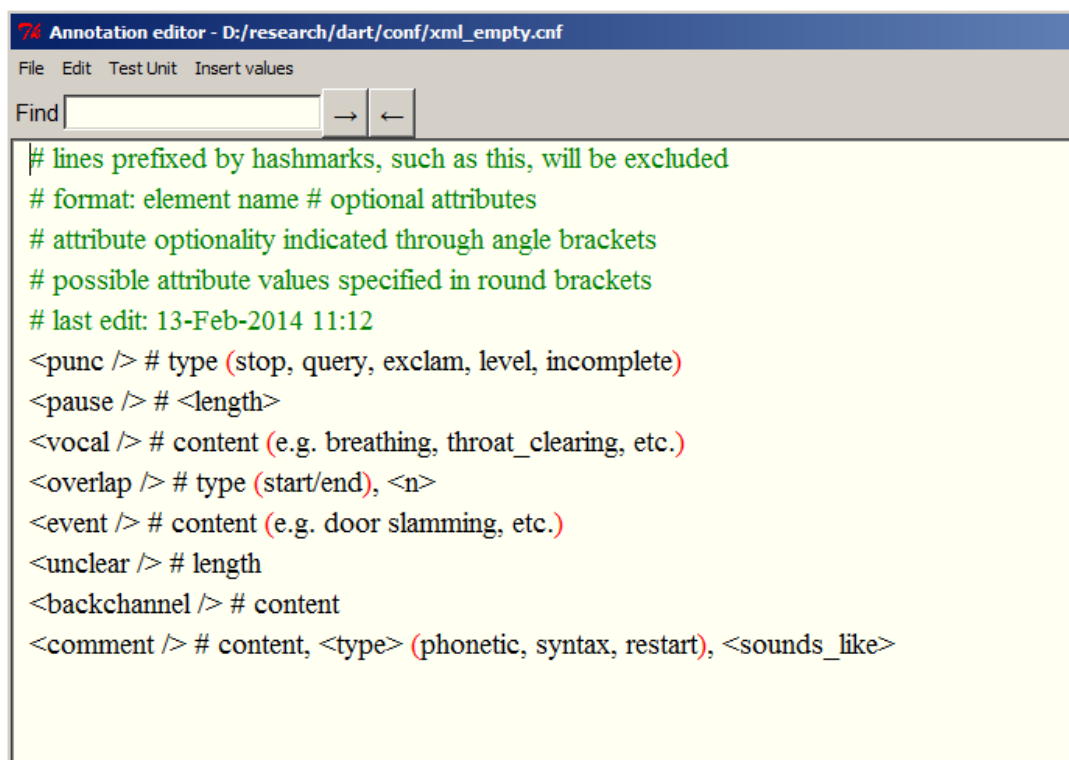
Figure 28 The editor displaying tag configuration sample

The above illustration shows the configuration file for empty XML elements opened for editing. The general format for all these configuration files is that the relevant element, attribute, value, or tag is listed at the beginning of the line, followed by a space, a hash mark[3], another space and then an explanation of the relevant item, which will later be displayed as a tooltip for the button. Additional items can freely be added or deleted here by the user, thus allowing different projects to create their own customised annotation options, as well as possibly constraining annotators/editors of the data to use only items that appear on the toolbar, while providing appropriate quick reference options via the tool tips.

---

[3] Pound symbol for those of you who speak American ;-).

8.2   Editing Semantico-Pragmatics, Semantics & Syntax Resource Files

The remaining menus, 'Edit modes file', 'Edit topics file', and 'Edit syntax file' and 'Add syntax file' from the 'Syntax' submenu, all provide access to facilities for editing or creating particular feature category definitions. *Modes* are semantico-pragmatic markers/key-phrases that indicate specific levels of verbal interaction that, on their own or in combination with syntactic features, assist DART in identifying speech acts. Thus, for instance an expression of a *condition*[4], unless it occurs inside a 'what if' question, signals that the speaker is conveying information that constrains the options provided in the context of the dialogue, in a similar way to how presenting an alternative (see below) 'widens' them. Using the little word *please*, where it is not a verb, clearly indicates that the speaker is making a(n indirect) request, even if the containing syntactic unit takes the form of an interrogative, while a 'discourse marker' like *I see* marks the utterer's *awareness* or understanding of what has been said, etc.

Topics, on the other hand, represent key-phrases that reflect the true semantic content of a syntactic unit, i.e. what is being talked about. Potential everyday (generic) topics that can be expressed as patterns, for instance, are phrases that indicate dates, times and time spans, durations, places, directions, etc. More domain-specific ones, e.g. in the Trainline context, would be patterns relating to bookings, particular types of tickets, seat options, journeys, rail cards/passes, etc. They are of slightly lesser interest to the pragmatic annotation side, but of course still provide valuable information that is worth investigating, especially if one wants to explore how particular topics (such as making appointments, etc.) are dealt with in a number of different dialogues, and by different speakers or speaker groups.

The modes, topics, and basic syntax files for turn-initial short units (such as discourse markers, yeses, nos, etc.), basically follow the same format. Each line that does not begin

---

[4] Mode labels in the examples below are given in italics.

with a comment symbol (#) specifies a label associated with a particular category, followed by a space double colon and a space (::). The remaining part of each line then contains one or more regular expression patterns, separated by three underscores (___). These patterns may minimally represent individual words, but unless these unmistakably represent a particular category, this could often be misleading, from either a computational or a conceptual linguistic perspective. For instance, on the computational side, the pattern that represents the mode *alternative* in the current implementation is defined as *either___\bor\b* (*\b* indicating a word boundary), where simply specifying *either___or* would intuitively seem correct to the average linguist who is not accustomed to computational analysis, but lead to fatal errors because not only would this quite rightly find *either*, *neither*, and *or*, but also *floor*, *flour*, *hour*, *poor*, *boring*, *organism*, etc., just because the latter words contain the character sequence <o> + <r>. Similarly, on the linguistic level, if we expected to be able to correctly identify all occurrences of *performatives* by simply specifying the performative marker *hereby* as a pattern, but ignoring that this ought to be preceded by a first-person pronoun, we could potentially end up with a number of false positives where some performative action is only being reported on, etc. Thus, writing appropriate patterns of this kind would minimally require a good understanding of regular expressions and an appropriate amount of linguistic expertise. However, the researcher who expects a particular pattern to be indicative of a specific phenomenon in the data could always at least test it directly on the data by creating the (sub-part of the) expression inside the concordancer and then pasting it into the corresponding resource file. This, in itself could potentially represent a good exercise for researchers who have little experience thinking in terms of linguistic patterns.

Regarding sub-categories for modes and topics, there is again an option to divide these into generic and domain-specific ones, although in general, modes tend to be generic in nature, and there is only one additional modes file so far, which specifies patterns that deal

with punctuation. This was actually incorporated into DART only at a relatively late stage because the analysis methodology was originally developed to work with files that excluded all punctuation, and also to some extent why the punctuation is not added directly to the text, but instead indicated by an empty XML element (e.g. *<punc type="stop" />*). The topic files, however, are clearly divided, and usually it is best to create a new domain-specific one for each corpus that is added, unless the new data is very similar to an earlier corpus. As the labels for modes and topics are also used to create value entries for post-editing tasks, in case it is impossible to define a particular pattern for a feature that is frequently needed, it is also possible to add an 'empty' label by simply writing the label name followed by the space, double colon, and space on a separate line in the resource file.

All the resource files discussed immediately above, plus the syntax files for short syntactic units, not only have the same basic syntax for defining patterns, but are also compiled at run-time into more complex regular expression patterns that make counting and adding these features to the annotation output easier. The main difference between modes and topics files, on the one hand, and 'short syntax' files, on the other, is that the latter are not subdivided into generic and domain-specific files, but rather organised into specific categories that are then checked for and removed from the beginnings of longer syntactic unit lines in a specified order and marked up as separate syntactic elements. In contrast, the other features are all added to the respective attribute labels according to their frequency of occurrence, so that the ordering inside the files is in fact irrelevant and can be used more to group related labels for conceptually similar categories together, where it is also advisable to add a comment line before the beginning of each group.

The remaining syntax files represent DART's 'grammars' for the major syntactic categories and, currently, editing the analysis routines unfortunately still requires considerable programming expertise and an in-depth understanding of almost all the processing mechanisms

involved, so tinkering with these is not for the faint-hearted. Further improvements to produce grammar files that would be easier to understand and manipulate for the 'average linguist' may be developed in future, though, in order to separate the linguistic logic behind the processing even further from that required for the programming part.

9   Editing Lexica

9.1   Basic Editing of Existing Lexica

Since morpho-syntactic analysis represents the first stage of the analysis process in DART, suitable lexicon files are an important prerequisite for achieving this task with a reasonable degree of accuracy. However, as no deep semantic – or even genuine full syntactic – analysis is performed in DART, these lexica can be kept quite simple and essentially consist of a *base form* entry, paired with a single morpho-syntactic tag, optionally followed by a brief description after a comment symbol, i.e. hash mark, as can be seen in Figure 29.
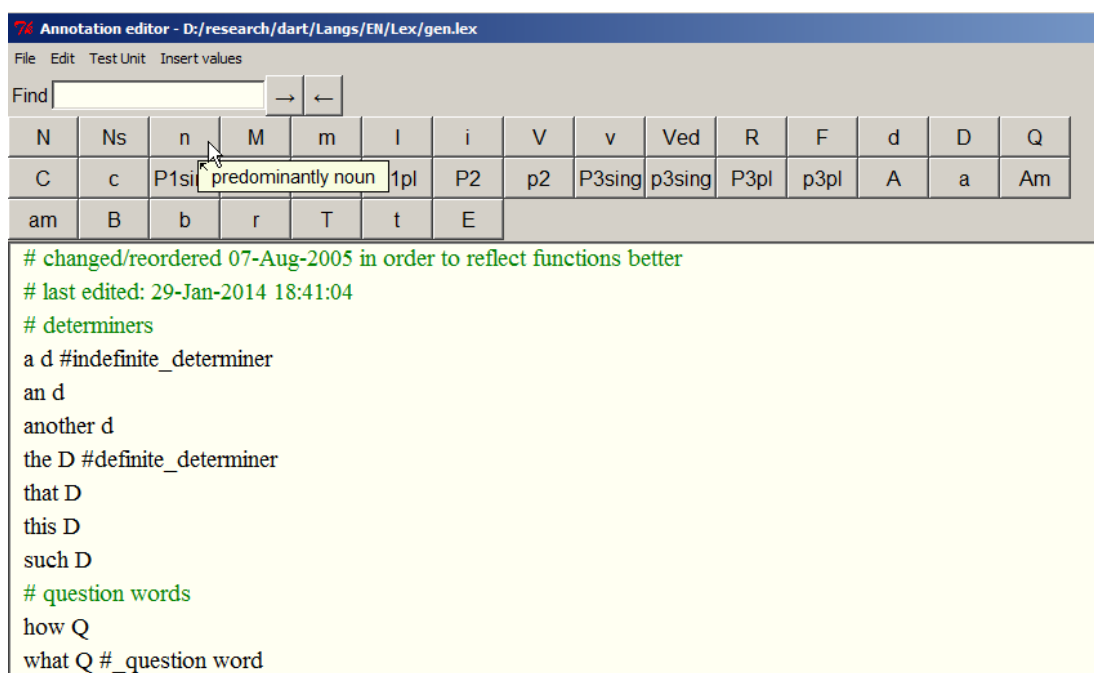

Figure 29 The editor displaying the generic lexicon ('gen.lex')

The tagset itself is also very limited, but has one very special feature: it encodes the potential for morpho-syntactic ambiguity[5] by making a distinction between upper- and lowercase tags. All tags currently used by DART are listed on toolbars when a lexicon is opened in the

---

[5]   … or *grammatical polysemy*, as I prefer to call it.

editor via the 'Lexica→Edit lexicon' option or the configuration editor interface, with a tool tip describing the meaning of each item, just like in the configuration files described above.

The main lexicon is the generic one (called 'gen.lex'), which contains some 400 base forms, and covers a majority of lexical items that occur in many – if not most – different types of dialogue, ranging from closed-class items, such as pronouns, auxiliaries, conjunctions, etc., to the most common content words. In general, it's best not to edit this lexicon, but usually create new, domain-specific ones, unless a word is definitely deemed to be generic enough for inclusion here.

Just to give a slightly better impresiion of what the coverage of the generic lexicon means in terms of actual token coverage: the word forms in this lexicon already account for between 70-80% of all tokens in the two main spoken corpora, Trainline (which deals with train information timetable and ticket bookings) & Trains (93; interactive goods shipment planning), that have primarily been analysed with DART so far, but even around 50% for written 1 Mio. word reference corpora, as verified using the LOB and the FLOB corpora. Bearing in mind that the entries in the generic lexicon are only base forms of words, it is easy to see that the token coverage can quite easily be enhanced by using a few simple morphological analysis routines.

The other option for improving lexical coverage is to add one or more domain-specific lexica. As stated above, 'dummy' files for new lexica can be created automatically through the corpus configuration editor, but the option described next is probably simpler in most cases.

9.2 Synthesising a New Lexicon

DART not only makes it possible to add and later include new lexica conveniently in the analysis process via its configuration options, but also takes advantage of the high coverage

achieved by the generic lexicon. It does this by providing the user with the option to create a frequency list from a new corpus which is automatically filtered by 'subtracting' items from the generic lexicon, again aided by some morphological analysis. This feature is triggered through the 'Synthesise lexicon' option and produces a result similar to the one depicted in Figure 30, where the first number represents a weighted frequency relative to the document frequency, whereas the second number indicates the raw frequency, i.e. total number of occurrences in the corpus.

Figure 30 A 'synthesised' lexicon, based on a data from the Amex corpus

In contrast to weighted values in information retrieval, such as *tf x idf*, which actually rank the word down if it has a high document frequency, assuming that it's less important for characterising according to distinct domains, the algorithm here instead uses a high document frequency as a positive feature and ranks the word higher, as it's deemed to be more important for the domain. The output then allows the researcher to 'prune' and save the list as a new lexicon by removing the frequencies displayed next to a 'synthesised' word form, which may of course still need to be reduced to its base form, and adding a tag from the PoS

toolbar. For completeness' sake, and to make the file conform with general format of lexicon files generated via the configuration editor, a file and corpus name should also be added in the header, in between the relevant place holders, marked >>> … <<<.

The mechanism behind the combination of generic lexicon and domain-specific lexica also provides a further advantage. Not only is it possible to add new vocabulary items in this way, but also to (re-)specify their default behaviour in terms of selecting for a particular part-of-speech. Taking the polysemous form *book*, for example, this could normally be used as both a noun or a verb, but would in general be more likely to be a noun. Hence, the generic lexicon would mark this particular ambiguity by using a lowercase tag *n* that indicates both that there is the inherent potential for grammatical ambiguity, but at the same time marks a preference for this word form to be used as a noun. In service- and task-oriented domains, in contrast, the word *book* is far more likely to be a verb, as in booking tickets, rooms, etc., though, so a domain-specific lexicon can simply over-write the default behaviour and change the tag to *v* instead, to indicate this preference, on the other hand still (hypothetically) allowing customers (e.g. in the Trainline data) to talk about particular seat preferences that permit them to read more leisurely. For grammatically unambiguous words, uppercase tags are used.

This approach has worked quite well so far, and also made it possible to devise highly flexible syntactic analysis routines that exploit this, but is obviously not without problems, partly because there are some rare cases where more than a two-way ambiguity is possible, and also because it requires the researcher to have a fairly intricate knowledge of all syntactic rules and how the aspect of ambiguity may influence them. Thus, a different approach may need to be developed in future implementations.

The 'Lexica' menu item also provides a facility for listing all words in a particular domain with a given morpho-syntactic tag or tags, specified as a regular expression, which is useful

for checking the coverage of a lexicon for particular parts-of-speech. This is done by choosing the 'Lexica→Show words by tag' option.

10 Known Issues

When 'installing' DART into a folder that contains Chinese characters, DART is unable to load configuration and/or data files. In this case, please install/copy the distribution into a folder without these characters in the path, e.g. C:\DART or D:\DART.

Avoid installing DART into any of the Windows 'Programs' folders, unless you have administrative rights and are prepared to change many permissions for sub-folders and configuration files. It appears that, by default, these folders do not allow the user to add or modify files, which is essential for running annotations or customising DART resources.