Annotating Communication Problems Using the MATE Workbench

Laila Dybkjær, Morten Baun Møller, Niels Ole Bernsen, Michael Grosse, Martin Olsen, Amanda Schiffrin

Natural Interactive Systems Laboratory,
Science Park 10, 5230 Odense M, Denmark
laila@nis.sdu.dk, baun@mip.sdu.dk, nob@nis.sdu.dk,
grosse@mip.sdu.dk, coma@mip.sdu.dk, mandy@nis.sdu.dk

Abstract

The increasing commercialisation and sophistication of language engineering products reinforces the need for tools and standards in support of a more cost-effective development and evaluation process than has been possible so far. This paper presents results of the MATE project which was launched in response to the need for standards and tools in support of creating, annotating, evaluating and exploiting spoken language resources. Focusing on the MATE workbench, we illustrate its functionality and usability through its use for markup of communication problems.

1. Introduction

The growing industrial take-up of language engineering products, and their constantly increasing variety and sophistication reinforces the need for tools and standards which can help making their development and evaluation more efficient. One aspect of this multi-faceted problem is the need for standardised annotated corpora and standard corpus annotation tools. In the case of spoken language dialogue systems (SLDSs), the need for tools and standards is evident as regards annotated spoken dialogue corpora and automatic information extraction. Information extraction from annotated corpora is used in SLDSs engineering for many different purposes, such as training and testing of components, constructing lexicons and grammars, extracting dialogue control structures, performing diagnostic evaluation of interfaces, and comparing and evaluating annotation schemes used by humans and/or machines.

The production of enriched corpus data is time- and cost-intensive. The idea of re-using annotated data is thus a very attractive one. So far, however, re-use of resources has usually required a painstaking, time-consuming and often inefficient adaptation process due to the lack of standards and widely used tools. Various initiatives have in recent years addressed the problem of markup standardisation, such as the Text Encoding Initiative (TEI), the Corpus Encoding Standard (CES), and (EAGLES). Whilst these initiatives have made progress on written language and/or current coding practice, none of them have focused on the creation of standards and tools for cross-level spoken language corpus annotation.

The European Telematics project Multi-level Annotation Tools Engineering (MATE) was launched in March 1998 in response to the need for standards and tools in support of creating, annotating, evaluating and exploiting spoken language resources. This paper provides an overview of the MATE project in Section 2. Section 3 briefly presents the workbench developed in MATE. Section 4 introduces the coding level for communication problems. Section 5 provides a thorough walkthrough of major workbench functionalities as seen from the user's point of view and illustrated through use of the workbench for communication problems markup. Section 6 concludes the paper by discussing future work and prospects.

2. The MATE Project

The aim of the MATE project has been to facilitate the re-use of spoken language resources by addressing theoretical issues as well as the practical implementation of solutions.

MATE has reviewed more than 60 existing annotation schemes relating to the coding levels addressed in the project, i.e. prosody, (morpho-)syntax, co-reference, dialogue acts, communication problems, and cross-level issues (Klein et al., 1998).

Based on the collected information and the consortium's experience, MATE has developed a standard framework for annotating spoken dialogue corpora at multiple levels, including those mentioned above (Dybkjær et al., 1998). The core concept of the MATE markup framework is that of a *coding module,* which extends and formalises the concept of a coding scheme. Roughly speaking, a coding module includes or describes everything that is needed in order to perform a certain kind of markup of a particular spoken language corpus. A coding module prescribes what constitutes a coding, including the representation of markup and relations to other codings.

The MATE markup framework has been used to ensure a common approach across the levels addressed. For each annotation level, one or more existing coding schemes were selected to form the basis of the best practice coding schemes to be included in the MATE workbench (Mengel et al., 2000). Common to the selected coding schemes is that these are among the most widely used coding schemes for their particular level, which means that they have been used by several annotators and for the annotation of many dialogues. All MATE best practice coding schemes are expressed in terms of coding modules.

Although the five coding levels addressed by MATE are very different, the MATE markup framework appeared to work well for all of them including their cross-level interactions. Use of the framework ensures a uniform description across levels of the best practice coding schemes. This enhances usability by making it easier for the annotator to move from one level to another and by facilitating use of the same set of software tools and ensuring the same interface look-and-feel across levels.

The MATE workbench is a corpus annotation toolbox which builds on the theoretical part of the MATE project discussed above. The following section presents the workbench (see also Section 5).

3. The MATE Workbench

A number of existing tools for annotating spoken dialogues were reviewed early in the project in order to provide input to the specification of the MATE workbench (Isard et al., 1998). Based on the specification, the MATE markup framework and the MATE best practice coding schemes, the functionality described below has been implemented as a java-based workbench, see (Isard et al., 2000) for implementation details. XML is used for internal representation of coding files but the user needs not know about XML to be able to use the workbench.

The MATE best practice coding schemes are included in the workbench as useful example coding modules and are ready for immediate use. In addition, users are offered the possibility of adding new coding schemes via the easy-to-use interface of the MATE coding module editor. One part of a coding module is a markup declaration. On the basis of the entered markup declaration, a DTD is automatically generated that defines which tags are available and how they can be used during markup of a corpus. The interface of the coding module editor builds on the MATE markup framework which serves as an intermediate layer ensuring that the user interface need not change even if the underlying XML representation of coding files should be changed and vice versa.

An audio tool offers the user the possibility of listening to speech files, e.g. while making a transcription, and to have sound files displayed as a waveform. A transcribed file may be annotated according to a selected coding module. The workbench itself only comes with a very simple transcription module. However, the workbench includes a bridge to the Transcriber tool (http://www.etca.fr/CTA/gip/Projets/Transcriber/) which means that dialogues transcribed with Transcriber can be used immediately for level markup in the MATE workbench.

A number of default style sheets define how output to the user is visually presented. Thus, for instance, phenomena of interest in the corpus may be given a certain colour or shown in boldface. The user may modify a style sheet or define new ones. At the moment, however, no style sheet editor is available, so a fairly detailed understanding of XSLT concepts and structure is required of those who want to write their own style sheets.

The workbench enables information extraction of any kind from annotated MATE corpora. In particular, any number of annotations from the level(s) marked up in the corpus can be combined in the query. Using the MATE query language editor, users can specify their query interactively, receiving support for consistency control. The result is shown as a set of references to the queried corpus. The query mechanism also supports extraction of statistical information from corpora (e.g. the number of marked-up nouns). Moreover, computation of important reliability measures, such as kappa values, is enabled.

Import of files from XLabels and BAS Partitur to XML format is supported. Other converters can easily be added to the workbench. Export to file formats other than XML may be achieved by using style sheets. For example, information extracted by the query tool could be exported to HTML in order to serve as input to a browser.

4. Communication Problems

One of the annotation levels addressed by MATE is that of communication problems. The increasing number of advanced SLDSs which support people in carrying out ordinary tasks, such as flight/train timetable consultation, ticket booking or directory inquiry, demands rigorous methods and tools for identifying, analysing, preventing and repairing problems in spoken human-machine interaction. Annotation of communication problems in spoken dialogue corpora can not only help developers and researchers extract information on the deficiencies of emerging dialogue interaction models, but can also yield clues as to how these might be improved.

Communication problems, if detected by users, typically lead to clarification or repair meta-communication. This is not really a problem in human-human dialogue. However, with current SLDS technology the possibility of real-time handling of clarification and repair meta-communication is seriously limited. User needs for clarification meta-communication that arise from the way the system addresses its domain, can easily surpass the system's meta-communication skills.

Nevertheless, detection of communication problems in SLDSs has so far usually been carried out on an ad hoc basis. It has generally been performed as a time-consuming task at a fairly late development stage if performed at all. In order to support a more cost-efficient development process for interaction models for SLDSs, we need a solid understanding of communication problems, both of their nature and why they occur. A straightforward way of approaching this problem is to annotate communication problems.

Communication problems annotation is still at an early stage, however. In the state-of-the-art review done in MATE, we only found one coding scheme which focused on communication problems. Many other schemes included some communication problem types but only because those problems were relevant to, e.g., dialogue act annotation which would then be the main focus of the coding scheme.

Communication problems are different in several respects from most other phenomena that are usually annotated and studied in a corpus. Most notably, communication problems need not necessarily be present in a corpus at all. In fact, the fewer there are, the better. This is in direct contrast with, e.g., prosodic and morpho-syntactic phenomena, or dialogue acts, which are present in any spoken dialogue corpus. To a large extent, the same is true for co-reference. All these phenomena are among the building blocks of spoken dialogue. Communication problems, on the other hand, are disruptive to a dialogue and co-operative human interlocutors usually try to avoid them. In particular for SLDSs, co-operative system communication is important for avoiding communication problems which often lead to user dialogue behaviour with which the system cannot cope.

The best practice communication problems coding scheme included in the MATE workbench, the Odense scheme, is based on a set of guidelines for co-operative spoken human-machine dialogue design (Bernsen,

Dybkjær and Dybkjær, 1998; Dybkjær, 1999). These guidelines have so far been shown to work for two-party, shared-goal human-machine dialogue. The primary focus of the Odense coding scheme is the markup of communication problems caused by the system because the emphasis is on investigating how system interaction can be improved to achieve a smoother dialogue with users. Of course, users also commit errors from time to time, which can be direct causes of communication problems. User errors have only been investigated to a limited extent in this context (Bernsen, Dybkjær and Dybkjær, 1998) and we still lack detailed knowledge of their mechanisms.

The set of tags (elements and attributes) used by the Odense scheme is small and simple, even if a three-component structure is involved, cf. the bottom of Figure 1. It is, however, a non-trivial task to identify communication problems and analyse them correctly to determine which guidelines they violate and how, i.e., which types of violation we are dealing with. Communication problems are tagged as types of violation of the guidelines for co-operative spoken dialogue. A particular guideline may be violated in several different ways. For example, GG7 (avoid ambiguity) would be violated by not saying whether the time "9 o'clock" given to the user by the system means 9 am or 9 pm. Another type of violation of the same guideline might occur if it was not made clear whether a certain flight arrival time refers to that given by the timetable or to the actual expected arrival time.

Such violation types are necessarily task dependent as they refer to concrete problems found in dialogues with particular applications. Thus, a communication problem refers both to the part of the orthographic transcription in which the guideline violation was found, and to a set of violation types which is being created along with the markup of communication problems. Each type of violation in its turn refers to the particular guideline that was violated.

For more information on how to detect and analyse communication problems using the cooperativity guidelines as a frame of reference, see (Dybkjær, 1999) and (Bernsen, Dybkjær and Dybkjær, 1998). These references include collections of examples of communication problems, violation types and references to the guidelines.

In the following we illustrate the main functionalities of the MATE workbench by describing how it is used for markup of communication problems.
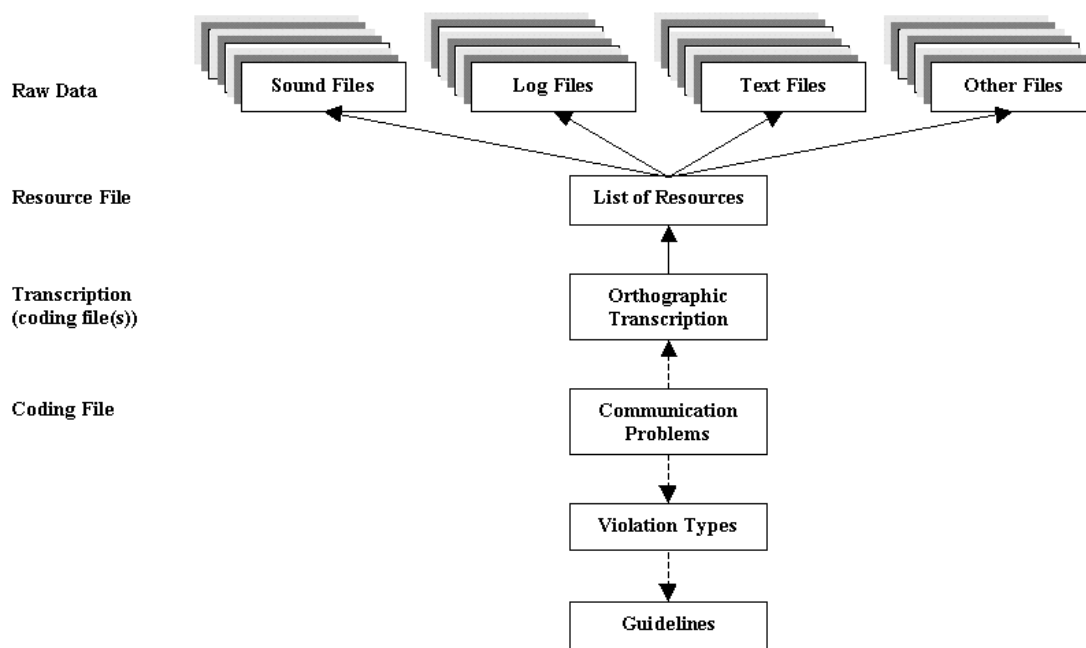

Figure 1. File organisation for a corpus annotated with respect to communication problems. The dashed arrow A --> B means that elements in A refer to elements in B by their attribute IDs, while the arrow A —> B means that there is a reference in A to B by its file name.

5. Markup of Communication Problems

When launched, the MATE workbench comes up with the two windows shown in Figures 2a and 2b. Figure 2a shows the controller window. Figure 2b shows a corpus folder window. The workbench tools can be opened from the controller window. The list of tools is extensible and is constructed automatically from the available set of tools. Status messages are written in the blank area under the menu. The corpus folder window is used for browsing, adding or editing corpus files. Additional corpus folder windows can be opened from the File menu of the controller window.
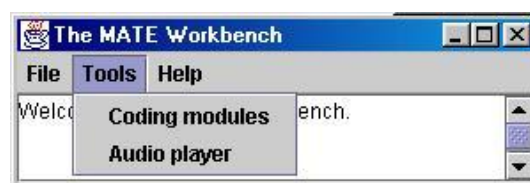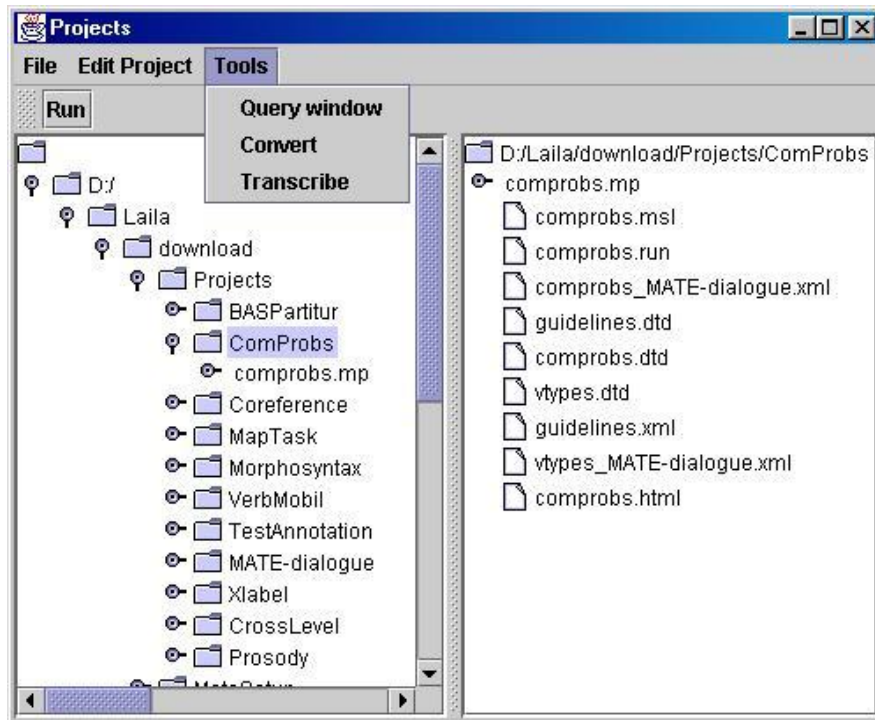

Figure 2a. The MATE controller window.

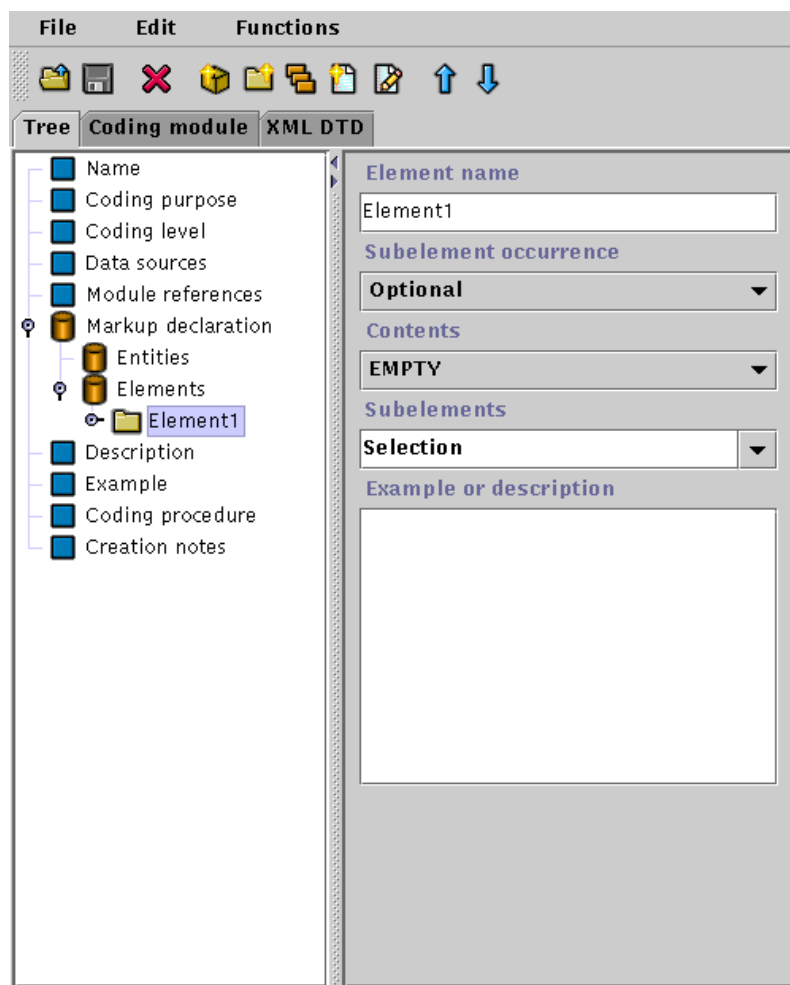**Figure 2b.** The corpus folder window.

Figure 3. The coding module editor.

The coding module editor (Figure 3) allows the user to enter new coding modules without knowing about XML which is used for internal file representation. Based on the element structure defined by the user, the tool itself automatically generates the XML document type definition (DTD) which is used internally in the MATE workbench.

The Odense coding scheme with its three coding modules is already included in the workbench as it is one of the MATE best practice schemes. To illustrate the concept of a coding module, the coding module for communication problems is shown in Figure 4. The coding modules for guidelines and violation types are not shown but follow the same principles and have the same list of entries.

Name: Communication_problems.
Coding purpose: Records the different ways in which generic and specific cooperativity guidelines are violated in a corpus. The communication problems coding file refers to a problem type coding file as well as to a transcription.
Coding level: Communication problems.
Data sources: Dialogue corpora.
Module references: Module Basic_orthographic_ transcription; Module Violation_types.
Markup declaration:
```
ELEMENT comprob
ATTRIBUTES
   vtype: REFERENCE(Violation_types, vtype)
   wref:  REFERENCE(Basic_orthographic_
          transcription, (w,w)+)
   uref:  REFERENCE(Basic_orthographic_
          transcription, u+)
   caused_by: REFERENCE(this, comprob)
   temp:  TEXT
ELEMENT note
ATTRIBUTES
   wref:  REFERENCE(Basic_orthographic_
              transcription, (w,w)+)
   uref:  REFERENCE(Basic_orthographic_
          transcription, u+)
```
Description: In order to annotate communication problems caused by inadequate system dialogue design we use the element comprob. It refers to some kind of violation of one of the cooperativity guidelines. The comprob element may be used to mark up any part of the dialogue which caused the communication problem. Thus it may be used to annotate one or more words, an entire utterance or even several utterances in which a communication problem was detected. The comprob element has five attributes.

The attribute vtype is mandatory. vtype is a reference to a description of a guideline violation type in a file which incrementally represents the different kinds of violations discovered of each individual guideline.

Either wref or uref must be indicated. Both these attributes refer to an orthographic transcription. wref delimits the word(s) which caused a communication problem, and uref refers to one or more entire utterances which caused a problem.

The attribute caused_by is optional. In some cases, a communication problem is caused by a problem which occurred earlier in the dialogue. caused_by is used to refer to a communication problem which was found elsewhere

in the dialogue and which led to the present communication problem.

The attribute temp is optional. It indicates a temporary markup. It usually takes a few dialogues before the coder gets a good grasp of the types of guideline violations which tend to occur in the corpus and what caused them. Often logfile inspection will be needed to make an exact causal diagnosis. Moreover, some problems become easier to detect when comparing several dialogues. Thus, temp is mainly for use during initial corpus markup but may also be used later if it is convenient to make temporary notes before making the final diagnosis. The vtype attribute overrides whatever communication problem the attribute temp indicates.

In the beginning of the analysis, the vtype attribute may be left open and the temp attribute filled in to describe the type of guideline violation identified. Very soon, however, a file containing the violation types should be established and in most cases the temp comments can simply be moved to this file and possibly modified to provide a violation type description. Note that due to this and to the coding procedure which requires at least two coders, the violation type references in the vtype attribute are likely to eventually be re-classified.

The note element can be used anywhere in a corpus to comment on whatever the user wants. It refers to one or more words or one or more utterances in the same way as the comprob element. The body of the note element contains text.

Example: The following example communication problems markup assumes a transcription from the Sundial corpus and refers to an example in the violation types coding module not shown in this paper:
```
<u id="S1:7-1-sun" who="S">flight information british
airways good day can I help you</u>
<comprob id="3" vtype="Sundial_problems#SG4-1"
uref="Sundial#S1:7-1-sun"/>
```
Coding procedure: We recommend to use the same coding procedure for markup of communication problems as for violation types since the two actions are tightly connected. As a minimum, the following procedure should be followed:
1. Encode by coders 1 and 2.
2. Check and merge codings (performed by coders 1 and 2 until consensus).

Creation notes:
Authors: Hans Dybkjær and Laila Dybkjær.
Version: 1 (25 November 1998), 2 (19 June 1999).
Comments: For guidance on how to identify communication problems and for a collection of examples, the reader is referred to (Dybkjær, 1999).
Literature: (Bernsen, Dybkjær and Dybkjær, 1998).

Figure 4. The communication problems coding module.

From the corpus folder window the user can select an already existing file to work on or create new files. It is possible to inspect the different types of file in a folder, cf. Figure 2b. For example, the user may inspect the DTDs generated by the coding module editor or view the style sheet used to display the coding file(s). Figure 5 shows a dialogue annotated using the Odense coding scheme and the default style sheet provided for communication problems.

Communication Problems

File Edit Tools Help

u	where are the people		
s	there are 2 groups of people at delta	GG1-1 GG3-1	N10
u	where are all the people		
s	there are 6 groups of people at delta and 1 at south delta	GG3-2	N11
u	use the helicopter to get the people from south delta to delta		N12
s	there are no people at south delta	GG3-2	N13
u	use the helicopter to get the people at south delta to delta		N14
s	your wish is my command	GG3-2 SG2-1	N15

GG1	Say enough.
SG1	State commitments explicitly.
SG2	Provide immediate feedback.
GG2	Don't say too much.
GG3	Don't lie.
GG4	Check what you say.
GG5	Be relevant.
GG6	Avoid obscurity.
GG7	Avoid ambiguity.
SG3	Ensure uniformity.
GG8	Be brief.
GG9	Be orderly.
GG10	Highlight asymmetries.
SG4	State your capabilities.
SG5	State how to interact.
GG11	Be aware of user background knowledge.
SG6	Be aware of user inferences.

N10	The only way in which this system phrase can be true is if there is a time delay and the system has not carried out the action asked for in utt_52 yet so that only the 2 groups of people from Exodus have arrived.
N11	The only way in which this system phrase can be true is if there is a time delay and the system has not finished carrying out the action asked for in utt_52 yet so that only the 2 groups of people from Exodus have arrived plus 4 of the groups from South Delta. Maybe the helicopter can only carry two groups at a time.
N12	Although the user has already asked to use the helicopter get the people from South Delta to Delta, the action has apparently not been carried out (utt_57) or has only to a certain extent been completed (utt_59), so the user now repeats his request.
N13	The only way in which this system phrase can be true is if

GG1-1	Insufficient information provided to the user on where the different groups of people are located. It is not clear why only two of the seven groups are mentioned.
GG3-1	If the system has carried out the actions which the user has asked for, there should be 7 groups of people at Delta rather than only those mentioned by the system.
GG3-2	The system provides information which is inconsistent with the information it provided in its previous utterance.
SG2-1	There is no feedback by the system to indicate that it has understood the user's prior utterance (request, action-directive, etc.). The utterance should be reflected back at the user in the response to reassure them that they are being correctly understood.
SG2-2	It is usually not a good idea to use coreferences in feedback situations because it does not become clear exactly what th

Figure 5. A dialogue annotated with communication problems.

MATE Audio Tool

File Help

Waveform

Current Position: 4.6375 seconds
Start Position: 3.425 seconds
End Position: 4.9625 seconds

Zoom

X Scale Factor: 0.01
Y Scale Factor: 25.0

Tag Clear Play Loop Stop

Figure 6. The audio tool.

In Figure 5 the dialogue is shown in the upper left-hand corner. The cooperativity guidelines are shown in shorthand version in the upper right-hand corner. Violation types are shown and can be added in the lower right-hand corner and notes in the lower left-hand corner.

During the detection and analysis of communication problems, an orthographic transcription of the dialogue is used. Often the logfile will have to be inspected as well, cf. the reference structure in Figure 1. In some cases it may even be necessary to access the sound files to, e.g., use intonation to disambiguate an ambiguous utterance in the orthographic transcription. For example, some questions have the same form as statements, and only the information provided by the intonation will reveal whether it is one or the other. The MATE workbench includes an audio tool which allows the user to play soundfiles, cf. Figure 6.

Having annotated a corpus the user may want to extract various kinds of information. The MATE workbench includes a query tool for this purpose, cf. Figure 7. It is one of the tools accessible from the tools menu in Figure 5. First of all, the user must select the document(s) to be queried. In a second step, the user can choose the element types to be included in the query expression from those available in the selected documents. Then the query expression can be built. Buttons in the interface are active as appropriate, and the attributes which belong to the selected element types are shown. Logical combinations and bracketing of simple query expressions can also be defined.

The result of a query is a document with a list of tuples of elements that are hrefs to the elements found. Figure 8 shows the result of asking for all types of cooperativity guideline violation found in a corpus. Since the output of the query is XML, the results can be displayed to the user in the same way as the data itself, using a stylesheet.
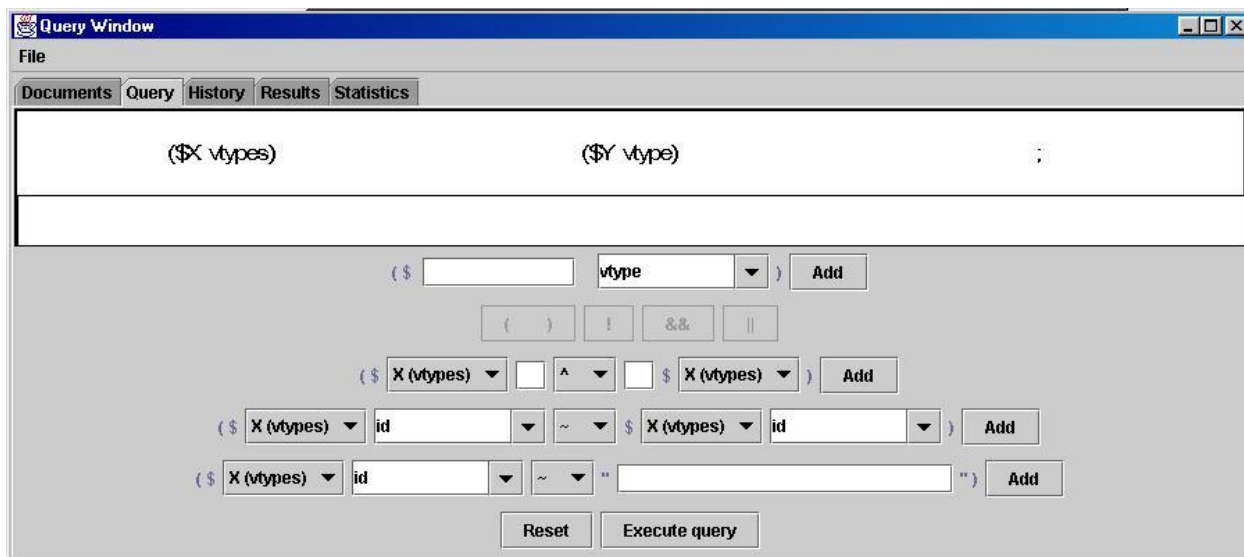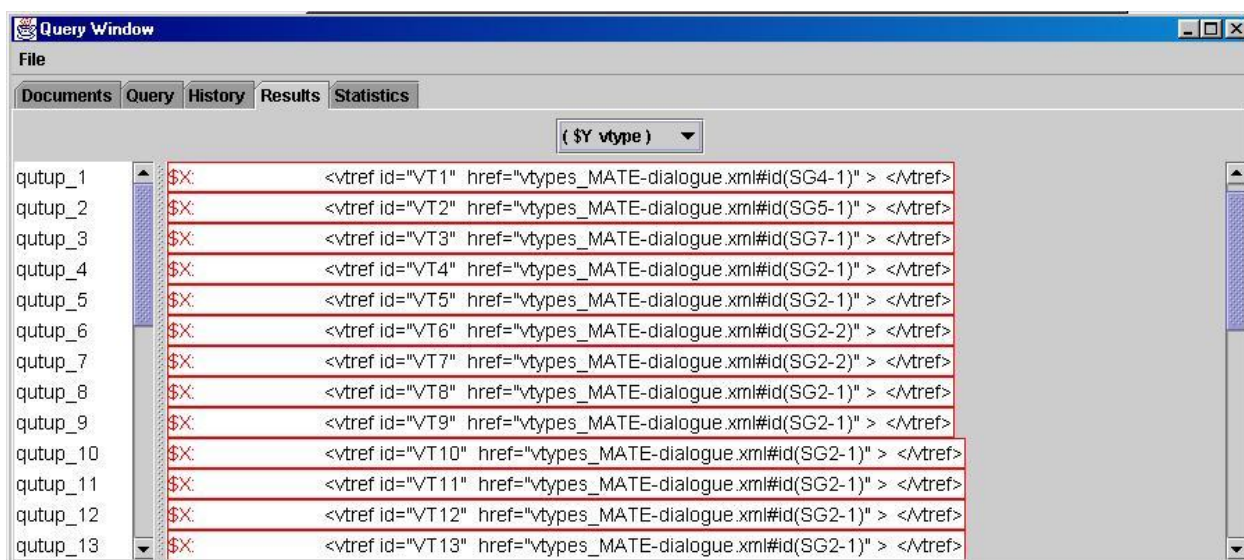

Figure 7. The query window.

Figure 8. Results of a query.

6. Future Work

A major aim of MATE has been to support the development of spoken language dialogue systems. The market for spoken dialogue systems is growing rapidly and so is the need for re-usable resources and for tools which facilitate their creation. MATE, it would seem, has therefore in many respects been timely and appropriate in responding to actual needs. This has also been apparent from the considerable interest shown in the MATE project world-wide.

The functionality of the MATE workbench is still being improved. The next generation of spoken dialogue systems are already taking first steps towards more natural interactivity by combining speech with other modalities, such as gesture and facial expression. As such multimodal dialogue systems are gaining ground, the need for re-usable multimodal resources as well as for tools and standardisation efforts in support of the development of multimodal systems, is increasing. A natural continuation of MATE would therefore be to re-use the successful MATE approach to the extent possible in order to create a markup framework and workbench for multimodal dialogue annotation.

7. References

Bernsen, N.O., Dybkjær, H. and Dybkjær, L., 1998. *Designing Interactive Speech Systems. From First Ideas to User Testing.* Springer Verlag.

Dybkjær, L., 1999. *CODIAL - a Tutorial and Tool in Support of Co-operative Dialogue Design.* http://www.disc2.dk/tools/codial/.

Dybkjær, L., Bernsen, N.O., Dybkjær, H., McKelvie, D. and Mengel, A., 1998. *The MATE Markup Framework.* MATE Deliverable D1.2.

Isard, A., McKelvie, D., Cappelli, B., Dybkjær, L., Evert, S., Fitschen, A., Heid, U., Kipp, M., Klein, M., Mengel, A., Møller, M.B. and Reithinger, N., 1998. *Specification of Workbench Architecture.* MATE Deliverable D3.1.

Isard, A., McKelvie, D., Mengel, A., Møller, M.B., Grosse, M. and Olsen, M.V., 2000. *Data Structures and APIs for the MATE Workbench.* MATE Deliverable D3.2.

Klein, M., Bernsen, N.O., Davies, S., Dybkjær, L., Garrido, J., Kasch, H., Mengel, A., Pirrelli, V., Poesio, M., Quazza, S. and Soria, S., 1998. *Supported Coding Schemes.* MATE Deliverable D1.1.

Mengel, A., Dybkjær, L., Garrido, J., Heid, U., Klein, M., Pirrelli, V., Poesio, M., Quazza, S., Schiffrin, A. and Soria, C., 2000. *MATE Dialogue Annotation Guidelines.* MATE Deliverable D2.1.

CES: http://www.cs.vassar.edu/CES/

EAGLES: http://www.ilc.pi.cnr.it/EAGLES/home.html

MATE: http://mate.nis.sdu.dk

TEI: http://etext.virginia.edu/TEI.html

Transcriber:
   http://www.etca.fr/CTA/gip/Projets/Transcriber/