The MATE Markup Framework

Laila DYBKJÆR and Niels Ole BERNSEN
Natural Interactive Systems Laboratory, University of Southern Denmark
Science Park 10, 5230 Odense M, Denmark
laila@nis.sdu.dk, nob@nis.sdu.dk

Abstract

Since early 1998, the European Telematics project MATE has worked towards facilitating re-use of annotated spoken language data, addressing theoretical issues and implementing practical solutions which could serve as standards in the field. The resulting MATE Workbench for corpus annotation is now available as licensed open source software.

This paper describes the MATE markup framework which bridges between the theoretical and the practical activities of MATE and is proposed as a standard for the definition and representation of markup for spoken dialogue corpora. We also present early experience from use of the framework.

1. Introduction

Spoken language engineering products proliferate in the market, commercial and research applications constantly increasing in variety and sophistication. These developments generate a growing need for tools and standards which can help improve the quality and efficiency of product development and evaluation. In the case of spoken language dialogue systems (SLDSs), for instance, the need is obvious for standards and standard-based tools for spoken dialogue corpus annotation and automatic information extraction. Information extraction from annotated corpora is used in SLDSs engineering for many different purposes. For several years, annotated speech corpora have been used to train and test speech recognisers. More recently, corpus-based approaches are being applied regularly to other levels of processing, such as syntax and dialogue. For instance, annotated corpora can be used to construct lexicons and grammars or train a grammar to acquire preferences for frequently used rules. Similarly, programs for dialogue act recognition and prediction tend to be based on annotated corpus data. Evaluation of user-system interaction and dialogue success is also based on annotated corpus data. As SLDSs and other language products become more sophisticated, the demand will grow for corpora with multilevel and cross-level annotations, i.e. annotations which capture information in the raw data at several different conceptual levels or mark up phenomena which refer to more than one level. These developments will inevitably increase the demand for standard tools in support of the annotation process.

The production (recording, transcription, annotation, evaluation) of corpus data for spoken language applications continues to be time-consuming and costly. So is the construction of tools which facilitate annotation and information extraction. It is therefore desirable that already available annotated corpora and tools be used whenever possible. Re-use of annotated data and tools, however, confronts systems developers with numerous problems which basically derive from the lack of common standards. So far, language engineering projects usually have either developed the needed resources from scratch using homegrown formalisms and tools, or painstakingly adapted resources from previous projects to novel purposes.

In recent years, several projects have addressed annotation formats and tools in support of annotation and information extraction (for an overview, see http://www.ldc.upenn.edu/-annotation/). Some projects have addressed the issue of markup standardisation from different perspectives. Examples are the Text Encoding Initiative (TEI) (http://www-tei.uic.edu/orgs/tei/ and http://etext.virginia.edu/TEI.html), the Corpus Encoding Standard (CES) (http://www.cs.vassar.edu/CES/), and the European Advisory Group for Language Engineering Standards (EAGLES) (http://www.-

ilc.pi.cnr.it/EAGLES96/home.html). Whilst these initiatives have made good progress on written language and current coding practice, none of them have focused on the creation of standards and tools for cross-level spoken language corpus annotation. It is only recently that there has been a major effort in this domain. The project Multi-level Annotation Tools Engineering (MATE) (http://mate.nis.sdu.dk) 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. The central idea of MATE has been to work on both annotation theory and practice in order to connect the two through a flexible framework which can ensure a common and user-friendly approach across annotation levels. On the tools side, this means that users are able to use level-independent tools and an interface representation which is independent of the internal coding file representation.

This paper presents the MATE markup framework and its use in the MATE Workbench. In the following, Section 2 briefly reviews the MATE approach to annotation and tools standardisation. Section 3 presents the MATE markup framework. Section 4 concludes the paper by reporting on early experiences with the practical use of the markup framework and discussing future work.

2 The MATE Approach

This section first briefly describes the creation of the MATE markup framework and a set of example best practice coding schemes in accordance with the markup framework. Then it describes how a toolbox (the MATE Workbench) has been implemented to support the markup framework by enabling annotation on the basis of any coding scheme expressed according to the framework.

2.1 Theory

The theoretical objectives of MATE were to specify a standard markup framework and to identify or, when necessary, develop a series of best practice coding schemes for implementation in the MATE Workbench. To these ends, we began by collecting information on a large number of existing annotation schemes for the levels addressed in the project, i.e. prosody, (morpho-)syntax, co-reference, dialogue acts, communication problems, and cross-level issues. Cross-level issues are issues which relate to more than one annotation level. Thus, for instance, prosody may provide clues for a variety of phenomena in semantics and discourse. The resulting report (Klein et al., 1998) describes more than 60 coding schemes, giving details per scheme on its coding book, the number of annotators who have worked with it, the number of annotated dialogues/segments/ utterances, evaluation results, the underlying task, a list of annotated phenomena, and the markup language used. Annotation examples are provided as well.

We found that the amount of pre-existing work varies enormously from level to level. There was, moreover, considerable variation in the quality of the descriptions of the individual coding schemes we analysed. Some did not include a coding book, others did not provide appropriate examples, some had never been properly evaluated, etc. The differences in description made it extremely difficult to compare coding schemes even for the same annotation level, and constituted a rather confused and incomplete basis for the creation of standard re-usable tools within, as well as across, levels.

The collected information formed the starting point for the development of the MATE markup framework which is a proposal for a standard for the definition and representation of markup for spoken dialogue corpora (Dybkjær et al., 1998). Analysis of the collected information on existing coding schemes as regards the information which came with the schemes as well as the information which was found missing, provided input to our proposal for a minimal set of information items which should be provided for a coding scheme to make it generally comprehensible and re-usable by others. For instance, a prescriptive coding procedure was included among the information items in the MATE markup framework despite the fact that most existing coding schemes did not come with this information. This list of information items which we call a coding module, is the core concept of the MATE markup framework and extends and formalises the concept of a coding scheme. The ten entries which constitute a coding module are shown in Figure 4. Roughly speaking, a coding module includes or describes everything that is needed in order to perform a certain kind of markup of spoken language corpora. A coding module prescribes what constitutes a coding, including the representation of markup and the relations to other codings. Thus, the MATE coding module is a proposal for a standardised description of coding schemes.

The above-mentioned five annotation levels and the issues to do with cross-level annotation were selected for consideration in MATE because they pose very different markup problems. If a common framework can be established and shown to work for those levels and across them, it would seem likely that the framework will work for other levels as well. For each annotation level, one or more existing coding schemes were selected to form the basis of the best practice coding modules implemented 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 respective levels in current practice, each having been used by several annotators and for the annotation of many dialogues. Since all MATE best practice coding schemes are expressed in terms of coding modules, they should contain sufficient information for use by other annotators. Their uniform description in terms of coding modules makes it easy for the annotator to work on multiple coding schemes and/or levels, and to compare schemes since these all contain the same categories of information. The use of coding modules also facilitates use of the same set of software tools and enables the same interface look-and-feel independently of level.

2.2 Tooling

The engineering objective of MATE has been to specify and implement a generic annotation tool in support of the markup framework and the selected best practice coding schemes. Several existing annotation tools were reviewed early on to gather input for MATE workbench specification (Isard et al., 1998). Building on this specification, the MATE markup framework and the selected coding schemes, a java-based workbench has been implemented (Isard et al., 2000) which includes the following major functionalities:

The MATE best practice coding modules are included as working examples of the state of the art. Users can add new coding modules via the easy-to-use interface of the MATE coding module editor.

An audio tool enables listening to speech files and having sound files displayed as a waveform.

For each coding module, a default stylesheet defines how output to the user is presented visually. Phenomena of interest in the corpus may be shown in, e.g., a certain colour or in boldface. Users can modify style sheets and define new ones.

The workbench enables information extraction of any kind from annotated corpora. Query results are shown as sets of references to the queried corpora. Extraction of statistical information from corpora, such as the number of marked-up nouns, is also supported. Computation of important reliability measures, such as kappa values, is enabled.

Import of files from XLabels and BAS Partitur to XML format is supported in order to demonstrate the usefulness of importing widely used annotation formats for further work in the Workbench. Similarly, a converter from Transcriber format (http://www.etca.fr/CTA/-gip/Projets/Transcriber/) to MATE format enables transcriptions made using Transcriber to be annotated using the MATE Workbench. Other converters can easily be added. Export to file formats other than XML can be achieved by using style sheets. For example, information extracted by the query tool may be exported to HTML to serve as input to a browser.

On-line help is available at any time.

The first release of the MATE Workbench appeared in November 1999 and was made available to the +80 members of the MATE Advisory Panel from across the world. Since then, several improved versions have appeared and in May 2000 access to executable versions of the Workbench was made public. The MATE Workbench is now publicly available both in an executable version and as open source software at http://mate.nis.sdu.dk. The Workbench is still being improved by the MATE consortium, so new versions will continue to appear. A discussion forum has recently been set up at the MATE web site where colleagues are invited to ask questions and provide information from their experience with the Workbench, including the new tools they have added to the MATE Workbench to enhance its functionality.

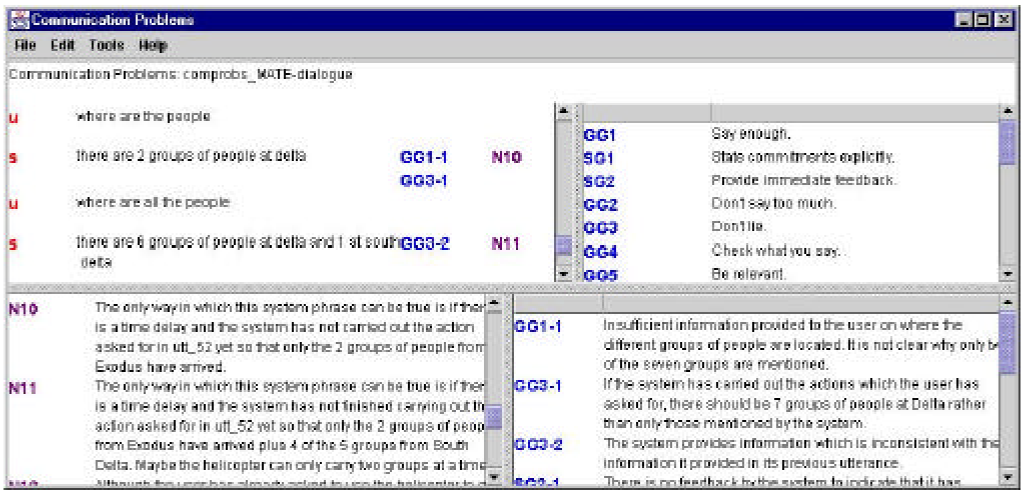
We have no exact figures on how many users are now using the workbench but we know that the MATE workbench is already being used by and is being considered for use in several European and national research projects.

3 The MATE Markup Framework

The MATE markup framework is a conceptual model which basically prescribes (i) how files are structured, for instance to enable multi-level annotation, (ii) how tag sets are represented in terms of elements and attributes, and (iii) how to provide essential information on markup, semantics, coding purpose etc.

3.1 Files, elements and attributes

When a coding module has been applied to a corpus, the result is a coding file. The coding file has a header which documents the coding context, such as who annotated the file, when, and the experience of the annotator, and a body which lists the coded elements. Figure 1 shows an example of how annotated communication problems are displayed to the user in the MATE Workbench. Figure 2 shows an excerpt of the internal representation of the file.



Communication Problems: comprobs_MATE-dialogue

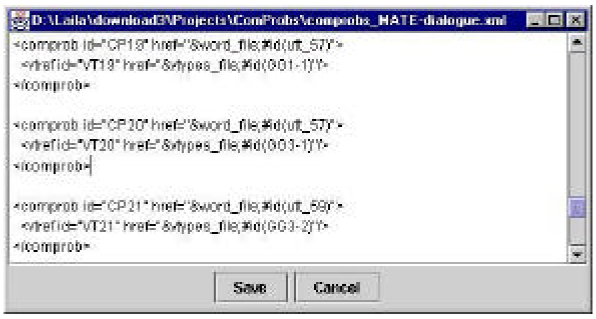
u	where are the people		
s	there are 2 groups of people at delta	GG1-1 GG3-1	N10
u	where are all the people		
s	there are 6 groups of people at delta and 1 at south delta	GG3-2	N11

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.

N10	The only way in which this system phrase can be true is if ther 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 ther is a time delay and the system has not finished carrying out th action asked for in utt_52 yet so that only the 2 groups of peop from Exodus have arrived plus 4 of the 5 groups from South Delta. Maybe the helicopter can only carry two groups at a time

GG1-1	Insufficient information provided to the user on where the different groups of people are located. It is not clear why only b 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.

Figure 1. A screen shot from the MATE Workbench showing a dialogue annotated with communication problems (top left-hand panel). Guidelines for cooperative dialogue behaviour are shown in the top right-hand panel. Communication problems are categorised as types of violations of the cooperativity guidelines. Violation types are shown in the bottom right-hand panel. Notes may be added as part of the annotation. Notes are shown in the bottom left-hand panel.


D:\Laila\download3\Projects\ComProbs\comprobs_MATE-dialogue.xml
Save
Cancel

```
<comprob id="CP19" href='&word_file;#id(utt_57)'>
 <vtref id="VT19" href="&types_file;#id(GG1-1)"/>
</comprob>

<comprob id="CP20" href='&word_file;#id(utt_57)'>
 <vtref id="VT20" href="&types_file;#id(GG3-1)"/>
</comprob>

<comprob id="CP21" href='&word_file;#id(utt_59)'>
 <vtref id="VT21" href="&types_file;#id(GG3-2)"/>
</comprob>
```

Figure 2. Excerpt of the internal XML representation of the annotated dialogue shown in Figure 1. The tags will be explained in Section 3.1.1 below.

As shown in Figure 2, the annotated file representation is simply a list of references to the transcription file. The underlying file structure idea is depicted in Figure 3 which shows how coding files (bottom layer) refer to a transcription file and possibly to other coding files, cf. entry 5 in the coding module in Figure 4. A transcription (which is also regarded as a coding file) refers to a resource file listing the raw data resources behind the corpus, such as sound files and log files. The resource file includes a description of the corpus: purpose of the dialogues, dialogue participants, experimenters, recording conditions, etc. A basic, sequential timeline representation of the spoken language data is defined. The timeline may be expressed as real time, e.g. in milliseconds, or as numbers indicating, e.g., the sequence of utterance starts.

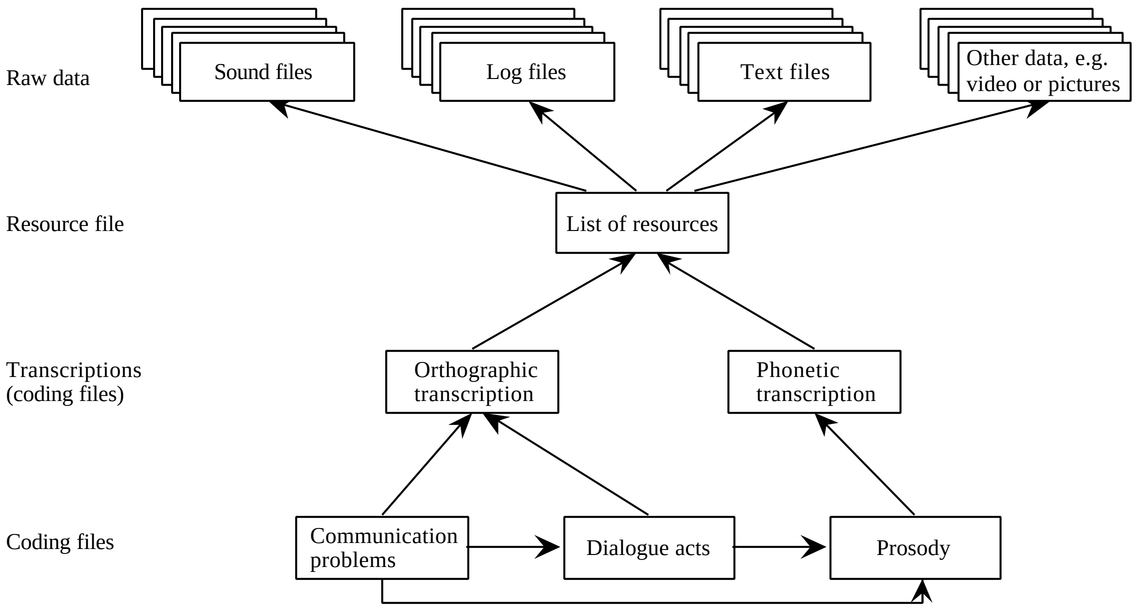


Figure 3. The raw corpus data are listed in the resource file to which transcriptions refer. Coding files at levels other than transcription refer to a transcription and only indirectly to the raw data. Coding files may refer to each other.

Given a coding purpose, such as to identify all communication problems in a particular corpus, and a coding module, the actual coding consists in using syntactic markup to encode the relevant phenomena found in the data. A coding is defined in terms of a tag set. The tag set is conceptually specified by, and presented to, the user in terms of elements and attributes, cf. entry 6 in the coding module in Figure 4. Importantly, workbench users can use this markup directly without having to know about complex formal standards, such as SGML, XML or TEI.

3.1.1 Elements

The basic markup primitive is the element (a term inherited from TEI and SGML) which represents a phenomenon such as a particular phoneme, word, utterance, dialogue act, or communication problem. Elements have attributes and relations to each other both within the current coding module and across coding modules. Considering a coding module M, the markup specification language is described as:

- $E_1 \dots E_n$: The non-empty list of tag elements.
- For each element E_i, the following properties may be defined:

1. N_i: The name of E_i.
   Example: <u>
2. E_i may contain a list of elements E_j from M.
   Example: <u> may contain <t>: <u><t>Example</t></u>
3. E_i has m_i attributes A_{ij}, where $j = 1 .. m_i$.
   Example: <u> has attributes who and id, among others.
4. E_i may refer to elements in coding module M_j, implying that M references M_j.
   Example: a dialogue act coding may refer to phonetic or syntactic cues.

A concrete example is the coding module for communication problems which, i.a., has the element <comprob>, cf. the XML representation in Figure 2. <comprob> has, i.a., the attributes id uref and vtype. uref is a reference to an utterance in the transcription coding. vtype is a reference to a type of violation of a guideline in the violation type coding. Due to the inflexibility of XML, this logical structure has to be represented slightly differently internally in the workbench. Thus, the uref corresponds to the first href in

Figure 2 while vtype is wrapped up in a new element and corresponds to the second href.

3.1.2 Attributes

Attributes are assigned values during coding. For each attribute A_{ij} the type of its values must be defined. There are standard attributes, user-defined attributes, and hidden attributes, as follows.

Standard attributes are attributes prescribed by MATE.

- id [mandatory]: ID. The element id is composed of the element name and a machine-generated number.
  Example: id=n_123

Time start and end are optional. Elements must have time information, possibly indirectly by referencing other elements (in the same coding module or in other modules) which have time information.

- TimeStart [optional]: TIME. Start of event.
- TimeEnd [optional]: TIME. End of event.

User-defined attributes are used to parametrise and extend the semantics of the elements they belong to. For instance, who is an attribute of element <u> designating by whom the utterance is spoken. There will be many user-defined attributes (and elements), cf., e.g., the uref and vtype mentioned above.

Hidden attributes are attributes which the user will neither define nor see but which are used for internal representation purposes. An example is the following of coding elements which may refer to utterances in a transcription but which depend on the technical programming choice of the underlying, non-user related representation:

ModuleRefs CDATA 'href:transcription#u'

See Figure 2 for a concrete example from the MATE Workbench.

3.1.3 Attribute standard types

The MATE markup framework proposes a set of predefined attribute value types (attributes are typed) which are supported by the workbench. By convention, types are written in capitals. The included standard types are:

- TIME: in milliseconds, as a sequence of numbers, or as named points on the timeline. Values are numbers or identifiers, and the declaration of the timeline states how to interpret them.

  Example: time=123200 dur=1280 (these are derived values, with time = TimeStart, and dur = TimeEnd – TimeStart).

- HREF[*MODULE, ELEMENTLIST*]: Here MODULE is the name of another coding module, and ELEMENTLIST is a list of names of elements from MODULE. When applied as concrete attribute values, two parameters must be specified:

  - The name of the referenced coding file which is an application of the declared MODULE coding module.
  - The id of the element occurrence that is referred to.

  The values of this attribute are of the form: “””CodeFileName”#”ElementId”””

  Example: The declaration OccursIn: href(transcription, u) allows an attribute used as, e.g., OccursIn=”base#u_123”, where base is a coding file using the transcription module and u_123 is the value of the id attribute of a u-element in that file.

  Example: For the declaration who: HREF[transcription, participant] an actual occurrence may look like who=”#participant2” where the omitted coding file name by convention generically means the current coding file.

  The concept of hyper-references together with parameters referencing coding modules (see point 5 in Figure 4) is what enables coding modules to handle cross-level markup.

- ENUM: A finite closed set of values.
  Values are of the form: “(“ Identifier (“|” Identifier)* “)”
  Example: time (year|month|day|hour) allows attributes such as time=day.
  The user may be allowed to extend the set, but never to change or delete values from the set.

- TEXT: Any text not containing ‘“’ (which is used to delimit the attribute value). *Example:* The declaration desc TEXT allows uses such as: <event desc=”Door is slammed”>.

- ID: Automatically generated id for the element. Only available in the automatically added attribute id.

3.2 Coding modules

In order for a coding module and the dialogues annotated using it to be usable and understandable by people other than its creator, some key information must be provided. The MATE coding module which is the central part of the markup framework, serves to capture this information. A coding module consists of the ten items shown in Figure 4.

1. Name of the module.
2. Coding purpose of the module.
3. Coding level.
4. The type of data source scoped by the module.
5. References to other modules, if any. For transcriptions, the reference is to a resource.
6. A declaration of the markup elements and their attributes. An element is a feature, or type of phenomenon, in the corpus for which a tag is being defined.
7. A supplementary informal description of the elements and their attributes, including:
   a. Purpose of the element, its attributes, and their values.
   b. Informal semantics describing how to interpret the element and attribute values.
   c. Example of each element and attribute.
8. An example of the use of the elements and their attributes.
9. A coding procedure.
10. Creation notes.

Figure 4. Main items of the MATE coding module.

Some coding module items have a formal role, i.e. they can be interpreted and used by the MATE workbench. Thus, items (1) and (5) specify the coding module as a named parametrised module or class which builds on certain other predefined modules (no cycles allowed). Item (6) specifies the markup to be used in the coding. All elements, attribute names, and ids have a name space restricted to their module and its coding files, but are publicly referrable by prefixing the name of the coding module or coding file in which they occur. Other items provide directives and explanations to users. Thus, (2), (3) and (4) elaborate on the module itself, (7) and (8) elaborate on the markup, and (9) recommends coding procedure and quality measures. (10) provides information about the creation of the coding module, such as by whom and when.

In the following, we show an abbreviated version of a coding module for communication problems to illustrate the 10 coding module entries.

Name: Communication_problems.
Coding purpose: Records the different ways in which generic and specific guidelines are violated in a given corpus. A communication problems coding file refers to a problem type coding file as well as to a transcription.
Coding level: Communication problems.
Data sources: Spoken human-machine 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 produced by inadequate system utterance design we use the element comprob. It refers to some kind of violation of one of the guidelines listed in Figure 1, top right-hand panel. The comprob element may be used to mark up any part of the dialogue which caused, or might cause, a communication problem. Thus, comprob may be used to annotate one or more words, an entire utterance, or even several utterances in which an actual or potential communication problem was detected. The comprob element has five attributes in addition to the automatically added id.

The attribute vtype is mandatory. vtype is a reference to a description of a guideline violation in a file which contains the different kinds of violations of the individual guidelines.

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

We stop the illustration here due to space limitations. The full description is available in (Mengel et al. 2000).

Example:

In the following snippet of a transcription from the Sundial corpus:

<u id="S1:7-1-sun" who="S">flight information british airways good day can I help you</u>

communication problems are marked up as follows:

<comprob id="3" vtype="Sundial_problems#SG4-1" uref="Sundial#S1:7-1-sun"/>

We do not exemplify note here and do not show the violation type coding file due to space limitations. However, note that once a coding module is specified in the MATE workbench, the user does not have to bother about the markup shown in the example above. The user just selects the utterance to mark up and then clicks on the violation type palette, or, in case it is a new type, clicks on the violated cooperativity guideline which means that a new violation type is added and text can be entered to describe it, cf. Figure 1.

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:

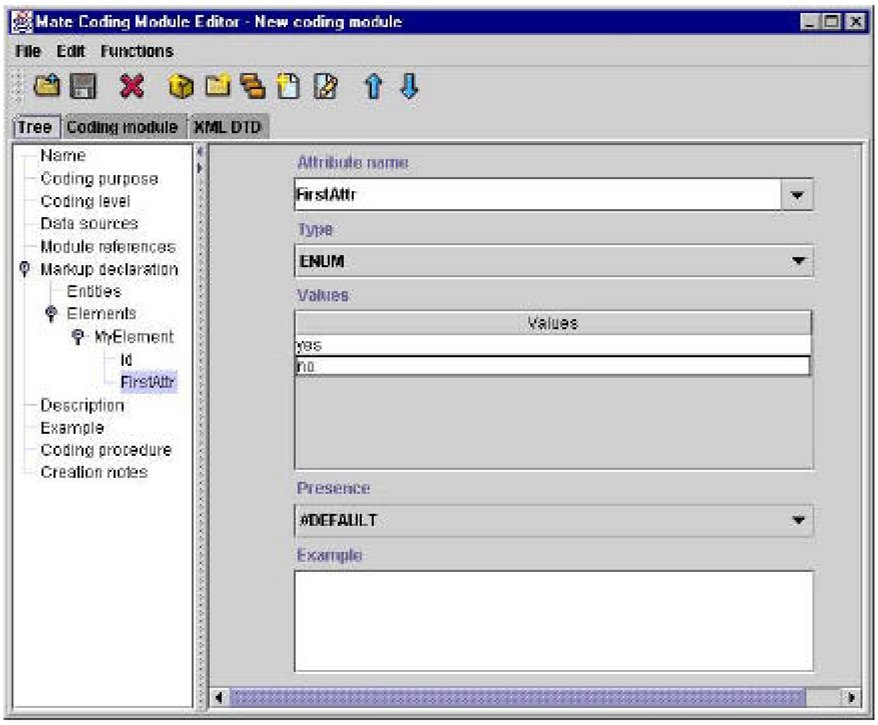
Authors: Hans Dybkjær and Laila Dybkjær.

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 invited to look at (Dybkjær 1999).

Literature: (Bernsen et al. 1998).

The MATE Workbench allows its users to specify a coding module via a coding module editor. A screen shot of the coding module editor is shown in Figure 5.



Attribute name: FirstAttr
Type: ENUM
Values: yes, no
Presence: #DEFAULT

Figure 5. The MATE coding module editor.

4 Early Experience and Future Work

The MATE markup framework has been well received for its transparency and flexibility by the colleagues on the MATE Advisory Panel. The framework has been used to ensure a common description of coding modules at the MATE coding levels and has turned out to work well for all these levels. We therefore conclude that the framework is likely to work for other annotation levels not addressed by MATE. The use of a common representation and a common information structure in all coding modules at the same level as well as across levels facilitates within-level comparison, creation and use of new coding modules, and working at multiple levels.

On the tools side, the markup framework has not been fully exploited as intended, i.e. as an intermediate layer between the user interface and the internal representation. This means that the user interface for adding new coding modules, in particular for the declaration of markup, and for defining new visualisations is still sub-optimal from a usability point of view.

The coding module editor which is used for adding new coding modules, represents a major step forward compared to requiring users to write DTDs. The coding module editor automatically generates a DTD from the specified markup declaration. However, the XML format used for the underlying file representation has not been hidden completely from the editor's interface. Peculiarities and lack of flexibility in XML have been allowed to influence the way in which users must specify elements and attributes, making the process less logical and flexible than it could have been. It is high on our wish list to repair this shortcoming.

As regards coding visualisation, XSLT-like style sheets are used to define how codings are displayed to the user. Writing style sheets, however, is cumbersome and definitely not something users should be asked to do to define how codings based on a new coding module should be displayed. We either need a style sheet editor comparable to the coding module editor as regards ease of use, or, alternatively, a completely new interface concept should be implemented to replace the style sheets and enable users to easily define new visualisations. It is high on our wish-list to better exploit the markup framework in the Workbench implementation in order to achieve a better user interface.

Other frameworks have been proposed but to our knowledge the MATE markup framework is still the more comprehensive framework around. An example is the annotation framework recently proposed by Bird and Liberman (1999) which is based on annotation graphs. These are now being used in the ATLAS project (Bird et al., 2000) and in the Transcriber tool (Geoffrois et al., 2000). The annotation graphs serve as an intermediate representation layer in agreement with the argument above for having an intermediate layer of representation between the user interface and the internal representation. Whilst Bird and Liberman do not consider coding modules or discuss the interface from a usability point of view, they present detailed considerations concerning time line representation and time line reference. The two frameworks may, indeed, turn out to complement each other nicely.

Acknowledgements

We gratefully acknowledge the support for the MATE project provided by the European Commission's Telematics/Language Engineering Programme. We would also like to thank all MATE partners. Without the very considerable joint efforts of the project consortium it would not have been possible to build the MATE Workbench.

References

Note on MATE deliverables: like the MATE Workbench, these are all obtainable from http://mate.nis.sdu.dk

Annotation formats and tools: http://www.ldc.upenn.edu/annotation/

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

Bird, S. and Liberman, M., 1999. *A Formal Framework for Linguistic Annotation.* Technical Report MS-CIS-99-01. Department of Computer and Information Science, University of Pennsylvania.

Bird, S., Day, D., Garofolo, J., Henderson, J., Laprun, C. and Liberman, M., 2000. ATLAS: A Flexible and Extensible Architecture for Linguistic Annotation. *Proceedings of the 2nd International*

Conference on Language Resources and Evaluation (LREC 2000), Athens, 1699-1706.

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

Dybkjær, L., 1999. CODIAL, a Tool in Support of Cooperative Dialogue Design. DISC Deliverable D2.8, April 1999. 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.

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

Geoffrois, E., Barras, C., Bird, S. and Wu, Z., 2000. Transcribing with Annotation Graphs. *Proceedings of the 2nd International Conference on Language Resources and Evaluation* (LREC 2000), Athens, 1517-1521.

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.

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

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.

TEI: http://etext.virginia.edu/TEI.html and http://www-tei.uic.edu/orgs/tei/

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