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

Keywords: markup framework, coding module, spoken dialogue corpora, multilevel annotation, usability.

Session: General.

Word count: 3199

Under consideration for other conferences: No.

Abstract: For two years, 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 is proposed as a standard for the definition and representation of markup for spoken dialogue corpora, and presents 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. This development generates a growing need for tools and standards which can improve the 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. Annotated speech corpora have been used for training and testing of speech recognisers for several years. More recently, corpus-based approaches are being applied regularly to other levels of processing, such as syntax and dialogue. Annotated corpora can be used, for instance, to construct lexicons and grammars or train a grammar to acquire preferences for frequently used rules. Similarly, programs for dialogue act recognition and prediction are 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 clearly grow for corpora with multilevel and cross-level annotations and for standard tools in support of the annotation process.

The production (recording, transcription, annotation, evaluation) of corpus data for spoken language applications is time-consuming and costly. So is the construction of tools which facilitate annotation and information extraction. Therefore, already available annotated corpora and tools should be used whenever possible. Re-use of annotated data and tools, however, confronts systems developers with numerous problems which basically refer to the lack of common standards. So far, language engineering projects usually have either developed the needed resources from scratch 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 of these 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/), 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 growing 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 and powerful 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

2.1 Theory

The theoretical objectives of MATE were to specify a standard markup framework and to identify, or develop, a series of best practice coding schemes for implementation in the MATE Workbench. To these ends, MATE started 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. 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 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). The core concept of the framework is the 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 spoken language corpora. A coding module prescribes what constitutes a coding, including the representation of markup and the relations to other codings.

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 schemes

proposed by MATE (Mengel et al., 2000). The MATE markup framework was then used to ensure consistency and a uniform approach across levels and schemes. Common to the selected coding schemes is that these are among the most widely used coding schemes for their level, having 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. This makes it easy for the annotator to work on multiple coding schemes and/or levels, facilitating use of the same set of software tools and providing 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 best practice schemes. Several existing annotation tools were reviewed early in the project to gather input to the MATE workbench specification (Isard et al., 1998). Building on the specification, the MATE markup framework and the best practice coding schemes, a java-based workbench has been implemented (http://mate.nis.sdu.dk, Isard et al., 2000), including the following major functionalities:

The MATE best practice coding schemes considered as working examples of the state of the art. Users can add new coding schemes 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 style sheet 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. 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.

3 The MATE Markup Framework

The MATE markup framework is a proposed standard for the definition and representation of markup in spoken dialogue corpora. The framework is a conceptual model which basically describes how files are structured, i.a. to allow for multi-level annotation, how tag sets are represented in terms of elements and attributes, and how to provide essential information on markup, semantics, coding purpose etc. by using coding modules.

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, when, and experience of the annotator, and a body which lists the coded elements. The general idea is depicted in Figure 1 which shows how coding files (bottom layer) refer to a transcription file and possibly to other coding files. 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. milliseconds, or as numbers indicating, e.g., the sequence of utterance starts.

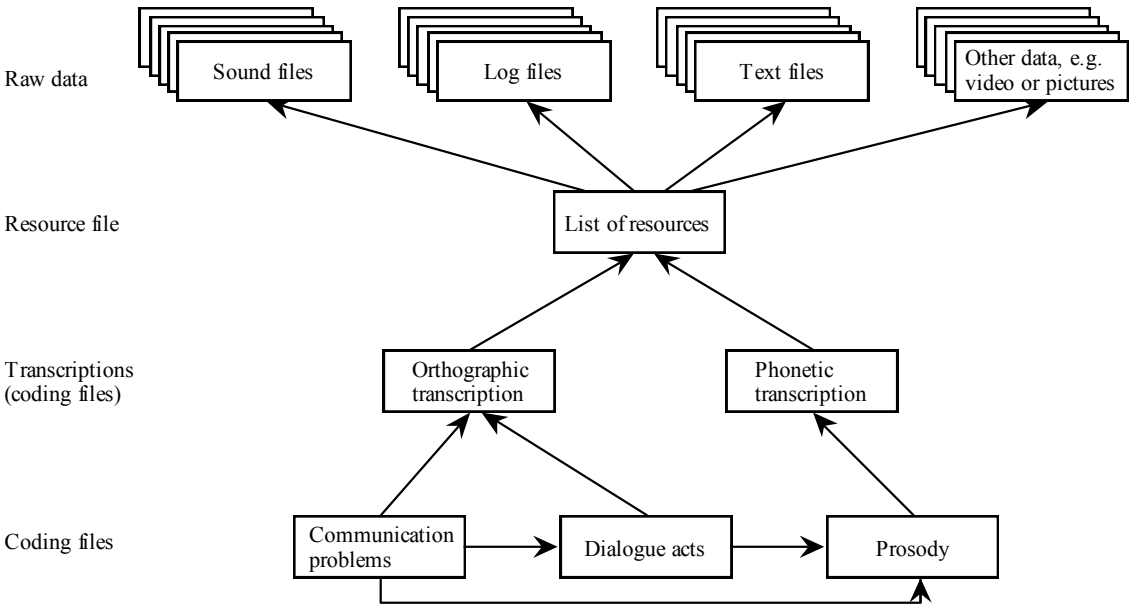


Figure 1. The raw corpus data are listed in the resource file to which transcriptions refer. Coding files at levels other than transcription refer to a transcription and only indirectly to the raw data. Coding files may refer to each other.

Given a coding purpose, such as to identify all proper names in a particular corpus, and a coding module, the actual coding consists in using syntactic markup to encode the 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. 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 ...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.

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 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. E.g., who is an attribute of element <u> designating by whom the utterance is spoken. There will be many user-defined attributes (and elements).

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'

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 2) 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 2.

1. Name of the module.

   Example: “Verbmobil dialogue acts”.

2. Coding purpose of the module.

   Example: “To code task-specific dialogue acts for Task T7”.

3. Coding level.

   Example: “Dialogue acts”.

4. The type of data source scoped by the module.

   Example: “Spoken dialogue corpora”.

5. References to other modules, if any. For transcriptions, the reference is to a resource.

   Example: “Orthographic transcription module OTM2 + Prosody module PM3 + Semantics module SM5”.

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 2. 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.

The final paper will provide more details on the MATE markup framework.

4 Early Experience and Future Work

The MATE markup framework has been well received for its transparency and flexibility by the +80 colleagues on the MATE Advisory Panel. The framework has been used to ensure a common approach to all coding levels and coding schemes addressed by MATE and appeared to work well in all cases. We therefore conclude that the framework is likely to work for other annotation levels and coding modules 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 sub-optimal from a usability point of view. The coding module editor is used for adding new coding modules. However, the XML format used for the underlying file representation has not been hidden well enough from the editor's interface. Peculiarities and lack of flexibility in XML have been allowed to influence the way the user must specify elements and attributes, making the process less logical and flexible than it could have been. 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. Therefore it is high on our wish-list to even better exploit the markup framework in the workbench implementation to achieve a better user interface.

Other frameworks have been proposed but to our knowledge the MATE markup framework is still the more comprehensive. 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 accordance with Section 3.1 above. Whilst Bird and Liberman do not consider coding modules or discuss the interface from a usability point of view, they present detailed considerations as regards time line representation and time line reference. The two frameworks may, indeed, turn out to complement each other nicely.

5 References

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

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., 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.

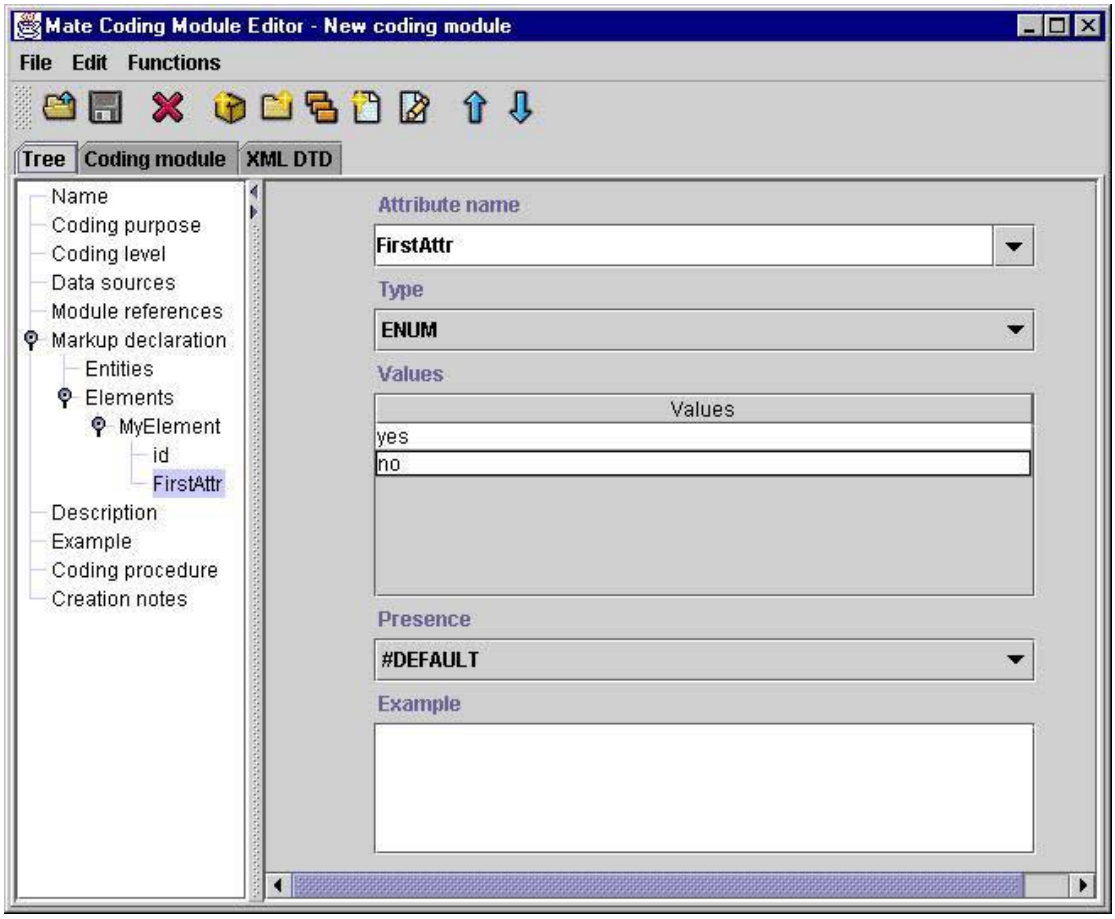
MATE: http://mate.nis.sdu.dk

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.

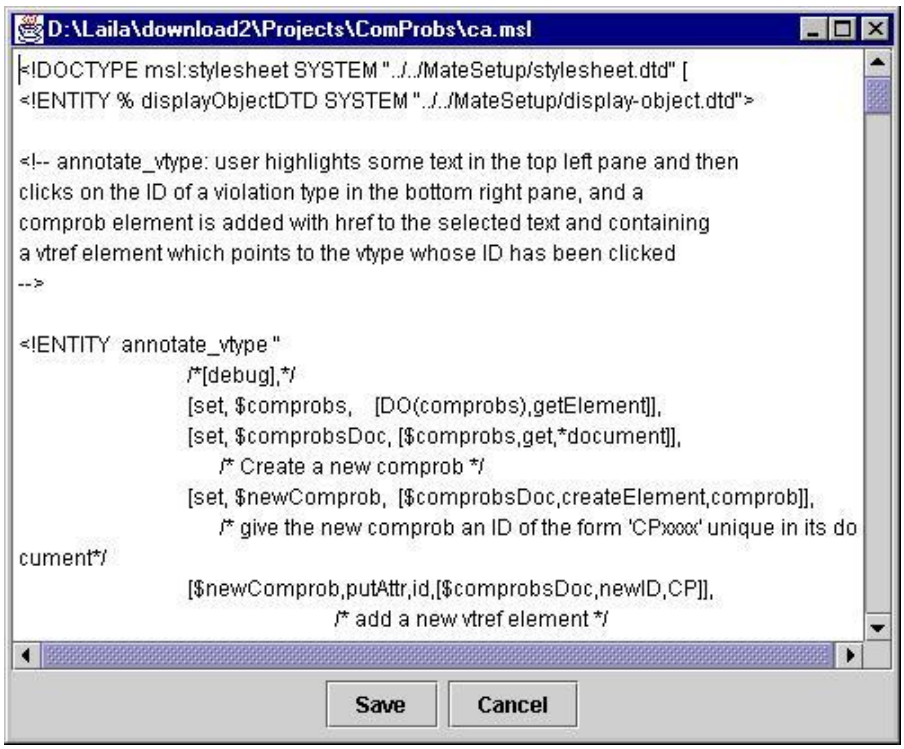
TEI: http://etext.virginia.edu/TEI.html

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

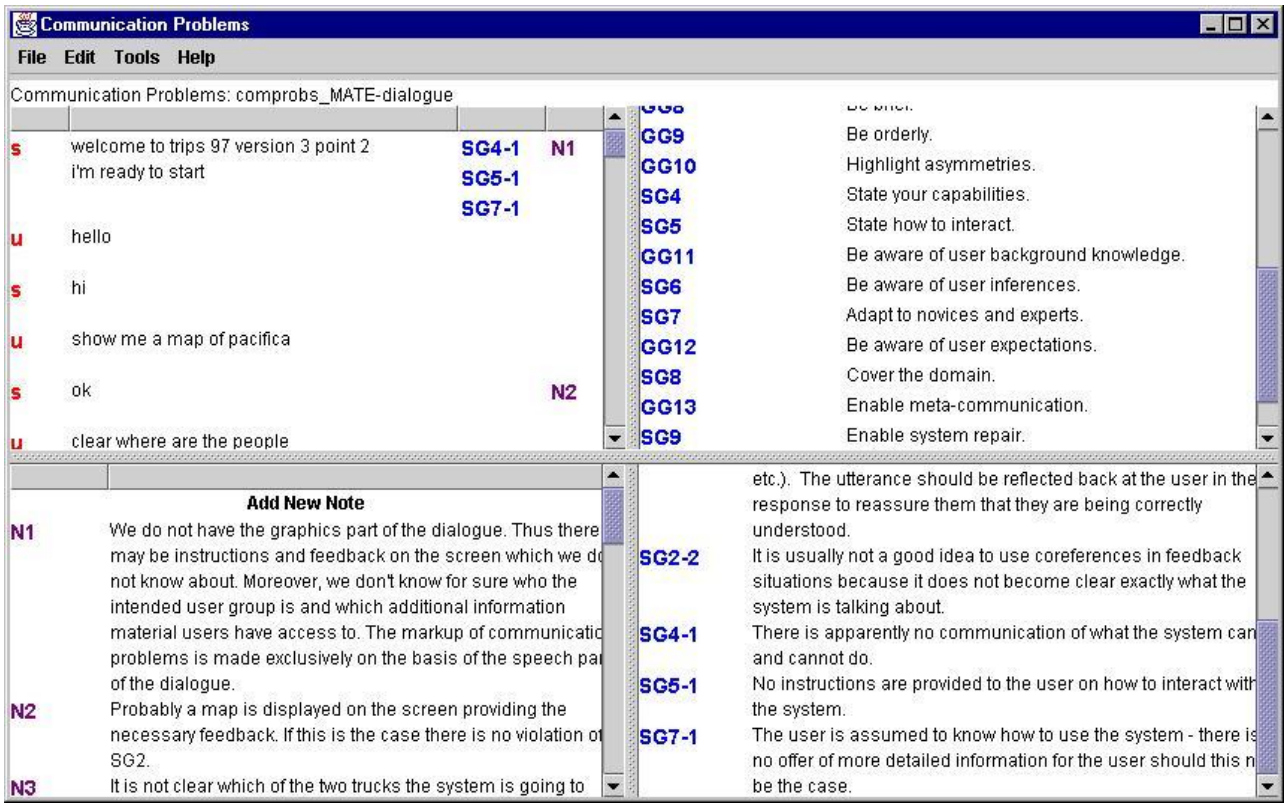
Appendix: Screen shots from the MATE workbench



The MATE coding module editor.



The style sheet for communication problems.



Visualisation using the style sheet for communication problems.