Project ref. no.	LE4-8370
Project title	MATE

Deliverable status	Public
Contractual date of delivery	30 November 1998
Actual date of delivery	30 November 1998
Deliverable number	D1.2
Deliverable title	The MATE Markup Framework
Type	RE
Status & version	Final
Number of pages	28
WP contributing to the deliverable	WP1
WP / Task responsible	Laila Dybkjær, MIP
Author(s)	Laila Dybkjær, Niels Ole Bernsen, Hans Dybkjær, David McKelvie, Andreas Mengel
EC Project Officer	Ray Hudson
Keywords	Markup framework, coding scheme, annotation, coding module, coding, transcription file, raw data, XML, TEI
Abstract (for dissemination)	This deliverable specifies a framework for the definition and representation of markup in spoken dialogue corpora. Focus is on the general issues involved in addressing standardisation at several levels of markup, including cross-level markup. The MATE markup framework is based on the notion of coding modules which extends and formalises the concept of a coding scheme. A coding module specifies the markup, its intended meaning, as well as the recommended coding process. A coding conforming to this specification is called a coding file of that module. A module is parameterised by other coding modules meaning that its markup may refer to markup elements of these other modules. All coding modules ultimately refer to markup of one or more transcriptions (e.g. phonetic or orthographic). A common timeline is used to align markup of codings that may otherwise be independent. The formats of coding modules and codings are specified in XML. Special consideration is given to the usability in terms of ease of use for coders.

MATE Deliverable D1.2
The MATE Markup Framework

30 November 1998

Authors

Laila Dybkjær[1], Niels Ole Bernsen[1], Hans Dybkjær[2],
David McKelvie[3], Andreas Mengel[4]

1: Natural Interactive Systems Lab, Odense University, Denmark. 2: Prolog Development Center A/S, Denmark.

3: HCRC, Edinburgh, UK. 4: Institut für Maschinelle Sprachverarbeitung, Stuttgart University, Germany.

Contents

1. Introduction

The aim of this MATE Working Paper is to discuss and, to the extent possible at this point in the MATE project, settle a series of issues that are fundamental to the development of MATE markup in Work Package 2 (WP2).

The type of linguistic resource addressed in the MATE project is spoken language corpora, in particular, spoken language dialogue corpora. Should further restriction of attention become necessary in MATE, focus will be on *task-oriented* spoken language dialogue corpora. A spoken language *corpus* is a body of spoken language data which has been recorded. A *spoken language dialogue* is a piece of speech communication between two or more participants where at least one participant is human.

A spoken language dialogue corpus is an inexhaustible source of information which can be used for many different purposes. For this to happen, the relevant information must be reliably marked up, starting with some form of transcription of the corpus. Following the markup of one or several groups of phenomena in the corpus, the marked-up corpus can be processed in many different ways in order to make use of the marked-up information for research, industrial or other purposes. And if the markup follows common standards, its re-use becomes much easier.

The present paper will discuss and propose solutions to the general issues involved in creating a universal standard for the markup of spoken language dialogue corpora. Based on the solutions proposed, the standard will be demonstrated, tested and further refined in actual corpus encoding practice in WP2. The possible coding schemes that can be used for markup purposes have been addressed in the MATE Working Paper D1.1 [Klein et al. 1998] and will be addressed again in WP2 where the details of marking up particular groups of corpus phenomena will be dealt with.

In Work Package 3 (WP3) a tool - the MATE Workbench - is being specified, designed and implemented to enable easy and fast coding according to the MATE standard as well as enabling efficient processing of the marked-up data. The MATE Workbench user interface specification is available as part of the MATE Working Paper D3.1 [Isard et al. 1998]. The main mechanisms for achieving ease of use, speed of use, and efficiency for the MATE Workbench are: (a) hiding for the user the technicalities of the coding representations so that the user can concentrate on coding contents (concepts, phenomena, tokens etc.), and (b) offering the user a series of useful and well-tested workbench functionalities. The MATE Workbench will not be described below but nevertheless lingers on the periphery of the discussion to follow, for instance in the discussion of the concept of a coding file.

2. Background

The MATE effort is not, of course, the first effort in markup standardisation for, or relevant to, spoken language dialogue corpora. In particular, MATE will build on and extend the following lines of work.

2.1 The TEI Guidelines

To the extent possible, MATE will use the *TEI* (Text Encoding Initiative) *Guidelines* [http://etext.virginia.edu/TEI.html] as a basis for markup. The TEI Guidelines provide a concrete set of *Standard Generalised Markup Language* (SGML) *Document Type Definitions* (DTDs) for the structured markup of any kind of text and make recommendations on their use. The Guidelines are designed to be applicable across a broad range of applications and disciplines and therefore address not only a vast range of textual phenomena, but are also designed for purposes of maximising generality and flexibility. The TEI standard is deliberately open-ended. The focus has been on defining a common encoding reference and guidelines for adding new (TEI) *compliant* or *conformant* DTDs in a way that allows for the interchange of annotated corpora. Generality and extensibility have been achieved at the cost of excessive complexity of the underlying representation. It is therefore a stated expectation of the Text Encoding Initiative that more specialised (less general) and simpler extensions will be specified for different corpus groups, and that support tools will be developed to enable easy and fast coding according to the encoding standard. MATE hopes to be able to satisfy these expectations.

TEI assumes the use of SGML which allows for fairly complicated DTDs. For markup languages this power is unnecessarily complex. This complexity is inherited by TEI where the wish to cater for any conceivable idea means that about all of the complexity of SGML is still present, with the additional layer of TEI. MATE would like to achieve a simpler and more direct markup declaration language and interface.

The TEI Guidelines include a proposal for the markup of orthographic transcriptions of spoken language [Sperberg-McQueen and Burnard 1994 Chapter 11]. The TEI Guidelines, however, do not go beyond spoken language orthographic transcription.

2.2 The CES and EAGLES

The *Corpus Encoding Standard* (CES, [Ide 1996, http://www.cs.vassar.edu/CES/]) is related to the *European Action Group for Language Engineering Standards* (EAGLES, [Gibbon et al. 1997, Leech et al. 1998, http://www.ilc.pi.cnr.it/EAGLES/ home.html]). The CES is an application to morpho-syntactic annotation of SGML compliant with the specifications of the TEI Guidelines. The CES thus provides an illustration of specialisation and extension of the TEI standard. The CES uses the TEI concept of modular DTDs and the TEI customisation mechanisms to select those pieces of the TEI proposals that are appropriate for corpus encoding. The CES specifies a minimal encoding level that corpora must achieve to be considered standardised in terms of descriptive representation (marking of structural and typographic information) as well as general architecture. It also provides encoding specifications for linguistic annotation, together with a data architecture for linguistic corpora.

From a MATE perspective, the main limitations of the CES are that the CES has mainly focused on written, non-dialogue corpora, such as newspaper texts, and does not go beyond the morpho-syntactic level. In many other respects, the CES is congenial to MATE.

In terms of contents, EAGLES go considerably beyond CES by looking at annotation of spoken dialogue corpora at the following levels: general coding issues, orthographic transcription, morpho-syntactic annotation, syntactic annotation, prosodic annotation, and pragmatic annotation including speech acts [Leech et al. 1998]. Thus, the aim of [Leech et al. 1998] is to survey current practices of annotation (or coding schemes) at those levels, and, when possible, identify those coding schemes that may be good models for others to follow. However, [Leech et al. 1998] do not go as far as proposing standards and do not assume the necessity of TEI conformance.

2.3 The Aims of MATE

In continuation of the standardisation and pre-standardisation work referenced in Sections 2.1 and 2.2 above, MATE will look at, i.a.:

- Standardisation at various (conceptual) levels of markup and their requirements, such as the need to represent non-hierarchical chain-like structures (typical for, e.g., referring expressions).
- Co-existence of different markups.

- Cross-level markup.
- Markup semantics.
- Coding procedures and quality of coding.
- Addressing mechanisms for referring to non-text data sources such as speech or video files.
- Additional requirements for "basic markup" of spoken language corpora, including issues such as transcription, the representation of overlapping speech (typical for, e.g., interruptions), background noise, etc.

3. The MATE Framework

This section introduces the core concepts of the MATE markup framework. These concepts are then expanded upon in the following sections. The concepts are:

- coding purpose;
- phenomena (or features), coding schemes;
- coding (or markup, or annotation);
- coding levels;
- cross-level coding;
- co-existing coding schemes;
- coding modules;
- coding procedures;
- coding files;
- basic coding, transcription, transcription files, timeline;
- raw data.

3.1 Coding Purpose, Phenomena, Coding Schemes, Coding

In general, markup, coding, or annotation assumes some particular *coding purpose,* such as to identify all proper names in a particular corpus. The number of possible coding purposes relative to some data source to be encoded, may normally be assumed to be unlimited. Given a coding purpose, the coding activity will focus on a series, or a group, of *phenomena* (or *features)* in the raw data, such as the proper names to be found in the data. Again, the number of possible groups of phenomena, or features, that might be coded in a particular data source, may be assumed to be unlimited. In MATE, any such group of phenomena that may be found in the raw data is called a *coding scheme.* Thus, in MATE, a coding scheme is a set of concepts of phenomena to be found in corpora. Given a coding purpose and a coding scheme, the actual *coding* consists in using some syntactic markup régime, such as a TEI conformant one, to encode the phenomena found in the data.

3.2 Coding Levels, Cross-level Coding

One guiding idea in MATE is that coding, markup, or annotation of spoken language dialogue and other spoken language corpora may be considered as being done at different *annotation levels,* such as orthographic transcription, prosody markup, or morpho-syntactic markup, as well as *cross-level annotation,* such as the markup of prosodic cues to speech acts. A level represents some level of abstraction at which selected information in the raw data can be identified, described, annotated, analysed and processed by machine. The levels addressed in MATE are:

- transcription;
- prosody;
- morpho-syntax;
- co-reference;
- speech acts;
- communication problems;
- cross-level coding.

MATE does not, of course, claim that the above levels are the only ones at which spoken language corpora can be usefully or even meaningfully annotated. Nor is there a claim that those levels can be simply and unambiguously defined. They merely serve as examples. Thus, MATE Workbench users may add annotation levels or modify existing annotation levels as they please. Still, the levels addressed in MATE are among the most important levels of abstraction at which spoken language corpora are being analysed in academic and industrial practice, reflecting the level of integration and sophistication of state-of-the-art interactive speech technologies [Bernsen et al. 1998].

Cross-level coding consists in representing various types of links (of co-existence, cueing relationship, cueing, etc.) between phenomena at different levels of annotation.

3.3 Co-existing Coding Schemes

Given the notions of (i) level of coding of spoken language corpora and (ii) coding scheme, it follows that several coding schemes may *co-exist*, complement one another or, in some cases, even compete at a certain annotation level or across levels. At the morpho-syntactic level of annotation, for instance, many different coding schemes co-exist and often the coding schemes compete as well - their competition being grounded in mutually exclusive theoretical claims to correctness, completeness etc. A large number of existing coding schemes belonging to the levels addressed in MATE, are presented in [Klein et al. 1998].

3.4 Coding Modules

A core notion in MATE is that of *coding module*. 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. The main parts of a coding module are:

1-4. Name, coding purpose, coding level, data source type.

5. References to other coding modules and to raw data resources.
6. A declaration of markup elements and their attribute names and types.
7. Supplementary informal description of the markup elements.
8. An example of the markup in use.
9. The recommended coding procedure.
10. Creation notes.

A complete description of the structure of a coding module is provided in Section 4.

New coding modules can be created either by defining a new coding module for an existing level or for existing levels (the cross-level case), or by defining a coding module that involves one or more new levels compared to the MATE levels.

3.5 Coding Procedures

A *coding procedure* is a recommendation for how corpus coding according to a particular coding module should be done in practice in order to be considered reliable. Coding procedures develop over time and it is in many cases difficult or impossible to determine which procedure is “the” correct one for a given coding purpose. Thus, several different coding procedures may well be “on offer” for a given coding purpose and coding scheme. Still, it is believed in MATE that, to the extent that these exist, guidelines for reliable coding are helpful to the users of MATE’s results. Note that, according to the terminology just introduced, two coding modules may be different from one another simply because of including different coding procedures; they can be identical in every other repect.

3.6 Coding Files

Coding consists in applying a coding module to a particular corpus. Each application of a coding module produces a single *coding file.* The coding file has a *header* which documents the parameters and context of the coding and a *body* which lists the coded elements. Note that there may well be several *versions* of a particular coding file to be indicated in the header, such as a half-finished version, a finished but non-proofed version, a proofed version which needs checking by a second person etc. Only the final, proofed version should be used for referencing by other coding files, as it is extremely important to maintain reliable, stable references. Coding files are described more fully in Section 5.

3.7 Basic Coding, Transcription

Given a spoken language dialogue corpus to be annotated, prior to annotation proper, so to speak, the corpus first has to undergo the standard process of *transcription.* The transcription presents who said what, in which order, events that might have happened during the spoken dialogue etc., and does so in a way which carries as little theoretical baggage as possible. The transcription rules are based on, for instance, standard dictionary word forms (orthographic transcription) or a standard sound alphabet (phonetic transcription). Following transcription, the transcribed corpus may be marked up, or annotated, in many different ways, at different levels, and following different coding schemes or, more generally speaking, different coding modules.

For this natural picture to be true, at least the following modifying remarks are required. First, transcription is also a form of coding, requiring a coding module of its own in order to be adequately defined. There is no obvious way to operationally define which information a certain form of transcription should or should not include. Depending on the coding or annotation purpose, a more or less detailed transcription will be needed. Secondly, there is no single obvious form of basic coding, or transcription, of spoken dialogue corpora. One candidate might be orthographic-transcription-at-word-level. However, a large class of annotation tasks is better served by taking phonetic transcription as the basic coding. In addition, sub-word level annotation may well need sub-word level transcription as its basic coding. Thirdly, it is at least conceivable that, for some coding purposes with respect to spoken language corpora, no form of corpus transcription at word level or sub-word level is needed whatsoever.

In summary, a *transcription* is a coding that refers to the raw data resources and to the timeline. Like any other coding the transcription is an application of a coding module, also called a *Transcription module*. A transcription will at times be called a *basic coding*. Issues of basic coding and transcription are addressed more fully in Section 6.

3.8 Transcription Files

MATE may need to provide coding modules for orthographic word level transcription, phonetic transcription, and, possibly, sub-word level transcription. However, none of the *transcription files* produced by applying one of the coding modules to the raw data of a corpus can be claimed to constitute "the" basic encoding of that corpus. In practice, however, other codings will refer to the raw data through one of these forms of transcription which further refer to raw data via a *resource file*. Thus, a transcription file provides an anchoring reference to the raw data, and this reference gets inherited by all other coding files which annotate the transcribed data. These other coding files are linked to the transcription file through hyperlinks between their body elements and the relevant elements, such as utterances, phrases, words or timeline points, in the transcription file. This allows for simple addition of new, coexisting codings of the corpus, all of which may refer to the same transcription file.

3.9 Raw Data, Resource File, Timeline

The transcription files normally provide reference to the raw data. Higher-level coding files will normally refer to raw data via the transcription file to which they hyperlink. These issues are further discussed in Section 7.

The raw data will be listed in a *resource file* which also contains a description of the corpus: What was the purpose, who participated in each session or dialogue, who were the experimenters, what were the recording conditions, etc. Whenever transcription files refer to raw data or persons, they do so through reference to the resource file.

In the resource file a basic, sequential *timeline representation* of the spoken language data is defined. This timeline may be in terms of real time, e.g. milliseconds, or in terms of ~~a~~ numbering indicating, e.g., the sequence of the utterance starts.

In general, each corpus transcribed and annotated according to the MATE standard will consist of the raw data files, the resource file, one or several basic coding files (or transcriptions), and any number of coding files hyperlinked to the basic coding files. Figure 1 provides an example.

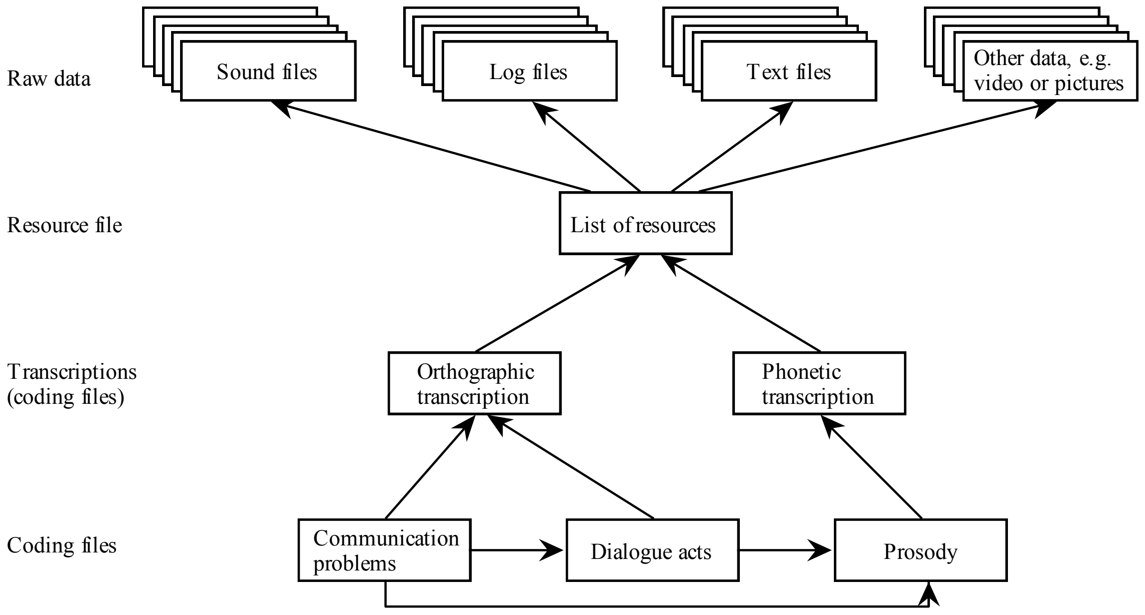

Figure 1. The raw data for a given corpus are listed in the resource file to which transcriptions refer. Coding files at levels other than transcription refer to a transcription and only indirectly to the raw data resources. Moreover, coding files may refer to each other.

The following sections will present in greater detail the MATE concepts that were briefly introduced above, as follows: coding modules (Section 4), coding files (Section 5), basic coding (Section 6) and raw data (Section 7). Section 8 addresses some implications for the MATE Workbench. Section 9 discusses the choice between SGML and XML. Section 10 concludes the Working Paper. Annex 1 presents a draft DTD for coding modules.

4. Coding Module

A coding module prescribes what constitutes a coding, including the representation of markup, and the relations to other codings. Users will be able to view the coding module and its corresponding coding files via an intuitive user interface in the workbench. Internally in the workbench, the coding module and coding files will be represented in XML providing an easy and precise parsing of the module and its corresponding coding files. In this section we provide an abstract, semi-formal definition of what constitutes a coding module. The user view is presented in Section 3.1 of deliverable D3.1 on the MATE workbench. The XML representation of coding modules is given in Annex 1.

4.1 Inventory of a Coding Module

A coding module consists of the following items:

1. Name of the module, including an acronym.
   Example: “Verbmobil dialogue acts [VM-DA]”.
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.

Some of the above items have a formal role (i.e. they can be interpreted and used by the MATE Workbench): 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 referable by prefixing the name of the coding module or coding file in which they occur. Other items in the inventory provide directives and explanations to the users: (2), (3), and (4) elaborate on the module level, (7) and

(8) elaborate on the markup level, and (9) prescribes the recommended coding procedure and quality measures. Item 10 provides information about the creation of the coding module, such as by whom and when.

4.2 The Coding Module

A coding module encapsulates the specification of a coding scheme. In terms of formal languages a coding module can be seen as an abstract type or class specification, exposing its element declarations to the world. Slots for commenting on coding purpose, coding level, and data source names are available.

Name

The name provides a title of the coding module, and an acronym used as in the file name for references to the module. The acronym must be unambiguous in the context in which it is used.

Parameters referencing coding modules

Together with the concept of hyperreferences to be explained under attribute standard types in the next section, this is what enables coding modules to handle cross-level markup.

Parameters referencing resources

Resource parameters are different from module parameters. Since we want to be able to re-use coding modules on several corpora we need the bottom level codings (i.e. transcriptions) to leave out the set of resources from the type check of module applications.

4.3 Elements and Attributes

A coding is defined in terms of a *tag set* with some structure. The tag set is conceptually specified by, and presented to, the user in terms of elements and attributes. Beneath the user interface these objects and their relations are represented in XML and may be transformed into TEI, CES or whatever encodings are supported for exchange of corpora. Users should be able to define markup and do coding in a straightforward way, which adds as little extra overhead as possible to the their analytical work and avoids the need to fight with markup technicalities. In the following, structures are defined that are conceptually simple and enable markup in a natural and direct way without forcing users to learn a complex formal language, such as SGML, TEI or XML, but instead letting them concentrate on their task.

Elements

The basic markup primitive is that of an *element* (a term inherited from TEI and SGML) that represents a certain type of 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 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.

Attributes

Attributes are assigned values during the coding, and 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.

The *standard attributes* are attributes the use of which is prescribed by MATE.

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

Whenever an element is inserted, the MATE Workbench adds the id attribute with a number which is unique relative to the coding and the element name. The underscore is necessary to separate suffix digits in the element name from the number.

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

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

Time start and end are optional in that they may not be needed for all elements. That is, elements must be anchored to the timeline, but this may happen indirectly by referencing other elements (in the same module or in other modules) that reference the timeline. Expectably, most codings will not worry about the timeline but reference it indirectly via a transcription.

The *user-defined* attributes are used to parametrise and extend the semantics of the elements they belong to. E.g., who is an attribute of elements <u> designating by whom the utterance is spoken. There will be a lot of user-defined attributes (and elements).

MATE has decided to include the elements and descriptive attributes of the package “TEI.Spoken” and make them a predefined recommendation for MATE users.

Finally, the *hidden attributes* are attributes that the user will neither define nor see but which are used for internal representation purposes by the MATE Workbench. Examples could be the following, but will depend on the final, technical programming choice of the underlying, non-user related representation:

```
TEIform CDATA ‘type’

ModuleRefs CDATA ‘href:transcription#u’

‘from:part_of_speech#pos’
```

where the first is an authentic example from a TEI DTD exposing technical details that users are forced to work with but should never have had to know about. The latter two are possible internal representations of (i) coding elements which may refer to utterances in codings of the transcription module, and (ii) the ‘from’ part of internal MATE support for references to intervals of the coded elements of a referenced module, here the part_of_speech module.

Attribute standard types

MATE will provide a set of predefined attribute value types (attributes are typed) that will have special support by the workbench. By convention, types are written in capitals. The included standard types are:

- TIME: in milliseconds, or as a sequence of numbers, or as references to 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).

The MATE Workbench may provide the option to operate in, e.g., seconds with one decimal for purposes of display and input. Example: time=123.2 dur=1.3

- 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.

- 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.

Two ways of thinking of and representing tag sets

A tag set must be able to mark occurrences of a phenomenon in a corpus and to characterise that phenomenon according to some taxonomy. For a given tag set one element marks the occurrences. However, the taxonomic characterisation of a tag set may be represented in two different ways using the above notation.

For tag sets with a *simple finite taxonomy*, the characterisations may be represented as the values of an attribute of type ENUM. An example is a coding scheme for dialogue acts with a finite set of dialogue acts.

For tag sets with more a *complex taxonomy*, the tag set is a combination of two elements. One element is used to mark the occurrence in the corpus with an attribute of type HREF linking to another element marking an (artificially added) list of compound characterisations. Example: in the coding of communication problems, the user compiles a current problem type list. This is well modelled via two elements: “problem_type” (describing a type of communication problem), and “problem” which has an attribute defined as: type:

HREF[communication_problems, problem_type]. If the characterisations have a more complex internal structure, it may be convenient to use several auxiliary elements.

The second way of representing tag sets using HREF and two elements is clearly needed and could be used as the only way to do it. However, for simple cases, MATE might allow the ENUM based one-element solution.

4.4 Coding Procedures

A coding procedure describes how the coding module should be applied to a corpus (or coding input) in order to produce a new coding. Basically, the importance of discussing, describing and comparing coding procedures is due to the fact that *reliable* codings are desirable. The coding procedure description should include:

1. Description of the coders: their number, roles and assumed training.
2. The steps to be followed in the coding.
3. Intermediate results, such as temporary coding files.
4. Quality measures (the non-satisfaction of which may require re-coding).

The coders are generically referred to as coder 1, coder 2, ... Occurrences of the same coder number indicate that the same coder should be used, and different coder numbers indicate that the coders must be different individuals.

The list of the actual coders will not exist until the coding module is being applied, at which point the coders are described in the coding file in terms of their role(s) in the coding procedure, their background, training, coding experience, and possibly other properties.

In the interest of promoting increased coding reliability, MATE proposes that coders should be able to express uncertainty wrt. to how to tag a particular token.

4.5 Example of a Coding Module

A XML DTD for coding modules is presented in Annex 1. An example of a coding module is shown in Figure 2, another in Figure 4.

The files involved in a coding module are:

a. *modulename*.cmd which contains the coding module description. ‘cmd’ is an acronym for ‘coding module’.

b. Temporary files which are all automatically generated by the workbench from the .cmd file, using in particular the (‘representation-independent’, i.e. independent of any external markup technical language used in coding files, such as XML or TEI) definitions of module references, elements, and attributes. For exchange purposes some of these files, like the *modulename*.dtd (a TEI dtd), may be externally available.

1. Module name: Word types (Type)

2. Module purpose: Records the type and abstract value of tokens. Uses one element <type>.

In the transcription one word is one token. Since the vocabulary is important in studies of human-computer spoken interaction, the domain specific tokens that come from a perhaps open range (e.g. numbers) or from a finite but incomplete range in the corpus (e.g. months) are marked to indicate the full set of possible values.

3. Coding level: Lexical.

4. Data sources: Spoken dialogue corpora.

5. Module references: Orthographic transcription module (OTM1).

6. Markup declaration:

```
ELEMENT   type
ATTRIBUTES
          type: ENUM  (month | name |  cardinal | ordinal)
          value: TEXT
          tokref: HREF(OTM1, token)
```

7. Description:

The following attribute values are possible for the "value" attribute, depending on the value of the "type" attribute:

month value ::= JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC.

name value ::= Person name. List is not finite and not enumerable.

cardinal value ::= 1 | 2 | 3 | ....

ordinal value ::= 1 | 2 | 3 | ... (i.e. represented by cardinal numbers).

A token not being one of the listed types is implicitly considered to have itself as a type, e.g the token 'rabat' is interpreted as if having the encoding <type type=rabat value="rabat">

8. Example:

<type type=month value="FEB" tokref="base:tok18">

Here assuming that the basic transcription is in the file "base.xml" and contains the token:

<token id="tok18">February</t>

9. Coding procedure:

1. Run the program "typenize.prl". This program will automatically encode all standard months, names, cardinals, and ordinals.

2. Let coder 1 manually read through the base transcription and mark names. Each part of a name should be marked separately as a name token, e.g. for "Peter van Deurs Jr." each of the four tokens should be marked with the type name.

3. Let coder 2 make a skim check of the coding.

Comment: The manual decision procedure in Step 2 is sufficiently robust for being judged by a single coder, and the check in Step 3 may be optional.

10. Creation notes:

Author: Hans Dybkjær.

Version: 3 (1998/11/01) [version 2 (1996/03/23), version 1 (1995/07/18)].

Comment: First created for use in the Danish Dialogue Project.

Literature: [Dybkjær et al. 1996].

Figure 2. Example of a coding module (adapted from [Bernsen et al. 1998, Figure 7.4]). Note how the attribute value has a value range depending on the type attribute value. This is only informally determined via the "semantics". In the coding procedure the coders are generically referred to as coder 1, coder 2, etc. Occurrences of the same coder number indicate that the same coder should be used. Different coder numbers indicate that the coders must be different individuals.

5. Coding Files

An application of a coding module follows a coding procedure and produces a coding file. Each coding file contains one coding only. A coding file consists of a *header* and a *body*. The header contains information on coding module and coding situation. The body contains the annotation.

The coding file *header* will include the information in the following (incomplete) list:

1. Date.
2. Version.
3. Name of coding module applied.
4. References to the actual coding files of the modules referred to in the coding module.
5. Names of the numbered coders in the coding procedure.

The coding in a coding file is represented in XML format. The coding file *body* will contain a list of the top-level elements (or tags) listed in the module declaration of elements. Only transcription coding files contain the transcription text, all other coding files refer to one such transcription file via hyperlinks. Any coding file can be hyperlinked to any other coding file and normally must be hyperlinked to a transcription file.

6. Basic Coding and Transcription

6.1 Timeline

The purpose of a timeline is to provide a common anchor point in time for the transcriptions and codings, and to sort out a sequentialisation of the markup. The resource timeline is presented in the body of the resource file and has the following declaration:

```
ELEMENT resourcetimeline  CONTAINS ( resourcept )
ATTRIBUTES
    Name : TEXT
    Id : ID
ELEMENT resourcept
ATTRIBUTES
    TimeUnit : ENUM( milliseconds, charnr )
    Type : ENUM( log, transcription, sound, video, image, notes )
    Id : ID
    TimeStart: TIME
    TimeEnd: TIME
    Res: HREF(this, resource)
```

where it is assumed that the resource file somewhere contains a list of element “resource” defining the resources. Note that in some cases one may have raw data in the corpus which have two different and incomparable sequences of events. In that case the resource file may contain two different timelines.

The transcription files will each contain a timeline defined as:

```
ELEMENT timeline CONTAINS ( pt )
```

```
ATTRIBUTES
    Name : TEXT
    Id : ID
ELEMENT pt
ATTRIBUTES
    ResPt : HREF( this, resourcept )
    Time : NUMBER
    id : ID
```

where Time is relative to the StartTime attribute of the referenced resource point ResPt. References to a timeline are made via a reference to a point pt on that line.

6.2 Two Basic MATE Coding Modules

A central form of basic coding of a spoken language corpus consists in coding, or aligning, corpus information relative to a timeline. Localisation of a particular phenomenon along a timeline serves to (in most cases) uniquely refer to that phenomenon as it occurs in the raw data file. In addition, timed units are fairly theory independent. Given the unlimited nature of potential coding purposes, it is conceivable that some MATE coding modules will lack even this basic timeline reference and merely rely on the sequential order of occurrence of the phenomena relevant to the coding purpose. Otherwise, according to MATE the timeline coding represents a basic property of a spoken language corpus.

It is an interesting fact that timeline coding of a spoken language corpus is not possible *per se*. *Something*, some information on the corpus, must be related to the timeline. In principle, this "something" could range from (i) the simple fact that the corpus runs from time 0 to time tn and thus has a duration of tn time units, to (ii) the exact mapping onto the timeline of each single piece of information contained in the corpus. Now, (i) is hardly of sufficient interest to require the creation of a MATE coding module, and it is impossible to create a coding module for (ii). MATE proposes to consider creating and testing coding modules for two basic codings involving timeline mapping, i.e. single word-level ("verbatim") orthographic transcription and phonetic transcription, whilst remaining open to the creation by future users of additional coding modules for transcription codings involving timeline mapping, such as sub-word level (e.g. morphological) coding, coding of gestures or other significant events accompanying the spoken language. The reason for calling the proposed MATE transcription codings *basic* codings is that the transcription coding files are likely to be referred to by many other coding files created according to MATE standards, i.e. the annotation files created on the basis of "higher" (non-transcription) level-specific coding modules. The exact specifications of the two MATE transcription coding modules will have to be done from the point of view of general usefulness for a range of purposes of higher-level annotation, i.e. through looking at the basic phenomena to which higher-level annotations need to refer.

Orthographic transcription:

Like all other coding files, orthographic transcription files consist of a header and a body. In addition to the information provided because of the fact that a transcription file is a coding file, the *header* includes information on (where applicable):

- Who transcribed; when (date/time).
- In which project.
- Contact details for obtaining additional information.

The *body* includes the transcription text divided into uniquely identified (numbered) utterances each of which is coded by unique reference to the speaker who produced it. Note that utterances are sometimes called *turns*, following the stereotype that people take turns in speaking. The default order of utterances in the basic transcription file will be according to time. In the orthographic transcription, the transcribed utterances contain uniquely identified word tokens. In addition, MATE will need to determine, in WP2, how to represent overlapping speech, such as through TEI time pointers, unfilled pauses and their relative lengths, filled pauses, false starts, repairs etc. ... spoken numbers, spoken word contractions, interjections, compounds, acronyms, abbreviations, word partials, ... unintelligible speech, laughter, non-vocal actions accompanying the dialogue, such as gesture, background noise etc., and how to use punctuation, capitalisation etc. For many of these phenomena, MATE will need to take into account the differences between the range of European languages addressed in the project. For some of the phenomena just mentioned, MATE may decide to let them be annotated, when needed, through higher-level coding modules, such as modules for part-of-speech tagging and syntactic tagging. All of these phenomena are related to the timeline. The time-alignment may follow one of two regimes:

1. Giving the time in milliseconds from recording start, indicating for each utterance or token the start time and the duration time.
2. A symbolic sequence enumeration analogous to the TEI <timeline>.

MATE recommends (1) as being the more robust solution for splitting and joining utterances or tokens, and because it automatically handles overlaps. However, (2) will be allowed because (1) is not always possible due to the fact that not all corpora are time stamped.

The MATE Workbench should provide the functionality for presenting the utterances in either of two formats:

1. As a vertical list sorted according to start time.
2. As a horizontal line for each participant, using vertical alignment as in a musical score.

Utterances and tokens should refer to raw data. An (incomplete) MATE coding module for orthographic transcription is shown in Figure 4 in Section 6.4. A series of EAGLES recommendations for a coding scheme for orthographic transcription are made in [Leech et al. 1998, 21-22].

Phonetic transcription:

The MATE approach to phonetic transcription will be developed in WP2 in connection with work on a standard for prosodic annotation. MATE work on phonetic transcription will be specifically designed to support prosodic annotation rather than to solve the myriad of problems involved in the development of more general-purpose coding modules for basic phonemic and phonetic transcription. The header of the phonetic transcription should include the following information:

- Who transcribed; when (date/time).
- In which project.
- Contact details for obtaining additional information.

6.3 Cross-level Basic Coding

Given the fact that MATE proposes to introduce two different basic transcriptions, cross-level issues arise already at this stage. In many cases, users may want to hyperlink the orthographic and phonetic codings of some corpus. In order to have some consistent (cross level) labelling and reference, the phonetic transcription might refer to the orthographic representation as follows:

A: Every word of the text is written down.

B: A standard normative phonetical representation is produced (by looking into a reference phonetic dictionary).

C: For each word, the actual pronunciation is transcribed, also describing the deviation from the original.

If the text is: “Das ist ein Beispiel” we have the words and their canonical transcription:

das: d a s

ist: ? I s t

ein: ? aI n

Beispiel: b aI S p i: l

The labels for the first two words would then be:

```
<phone start="12.123" end="12.223" canon="d">d</phone>
<phone start="12.223" end="12.323" canon="a">I</phone>
<phone start="12.323" end="12.434" canon="s">s</phone>
<phone start="12.434" end="12.434" canon="?">-</phone>
<phone start="12.434" end="12.543" canon="I">I</phone>
<phone start="12.543" end="12.656" canon="s">s</phone>
<phone start="12.656" end="12.736" canon="t">-</phone>
```

So in a canonical form it should be <das ist> [das ?Ist] but is pronounced as [dIsIs]. Sound deletions are marked by a hyphen (-) instead of a real element.

6.4 Inter-relating Timed Units

The <das ist> vs. [dIsIs] example provided at the end of Section 6.3 highlights the fact that true (to the raw data) timeline mapping at the word level is only possible, if at all, in phonetic transcription. Orthographic transcription will only be able to preserve true (to the raw data) timeline mapping at more coarse-grained levels. At the word level and still more fine-grained levels, the orthographic transcription will only preserve the sequential order-mapping into the raw data. Yet the sequential ordering is what matters for, e.g., part of speech tagging (morphosyntax) and syntactic description.

The following examples illustrate how issues of unit time labelling compound when taking higher levels of annotation into account. One might, for instance, want to POS (part-of-speech) tag sequences of ‘words’ together, such as “New York” or “you know” (as in “I’m going to the, you know, the pub”). Thus, the level of POS tagged units becomes different from the ‘word’ level which again may be different from the timed units of the basic orthographic level. In the case of “New York’s “, we would have:

Orthographic transcription:

```
<u who=A id=u1782 time=12082 dur=812>New York’s</u>
```

Word coding:

```
<word href=”base# u1782” id=w4011>New</word>
<word href=”base# u1782” id=w4012>York</word>
<word href=”base# u1782” id=w4013>’s</word>
```

Part of speech (POS) coding:

```
<pos href="base# u1782" id=p3987>New_York</pos>

<pos href="base# u1782" id=p3988>'s</pos>
```

This example is relatively straightforward. The following example illustrates how different types of operation must be defined in order to allow appropriate linking to the timed units:

Orthographic transcription:

```
<u who=B id=u1783 time=13008 dur=2505>
    Is New York <pause dur=455 time=13920>
    Police Department big?</u>

<u who=A id=u1784 time=13899 dur=478 >
    <vocal desc="cough"></u>
```

Word coding:

```
<word href="base# u1783" id=w4014>Is</word>

<word href="base# u1783" id=w4015>New</word>

<word href="base# u1783" id=w4016>York</word>

<word href="base# u1783" id=w4017>Police</word>

<word href="base# u1783" id=w4018>Department</word>

<word href="base# u1783" id=w4019>big</word>

<word href="base# u1784" id=w4020>_cough</word>
```

Part of speech (POS) coding:

```
<pos href="base# u1783" id=p3989>Is</pos>

<pos href="base# u1783" to="base#tu2009" id=p3990>
New_York_Police_Department</pos>

<pos href="base# u1783" id=p3991>Big</pos>
```

The POS tagging in the example requires the possibility of merging two different timed units into one POS unit. Other examples of operations on timed units are split, overlay, swap, delete and insert, cf. Figure 3.

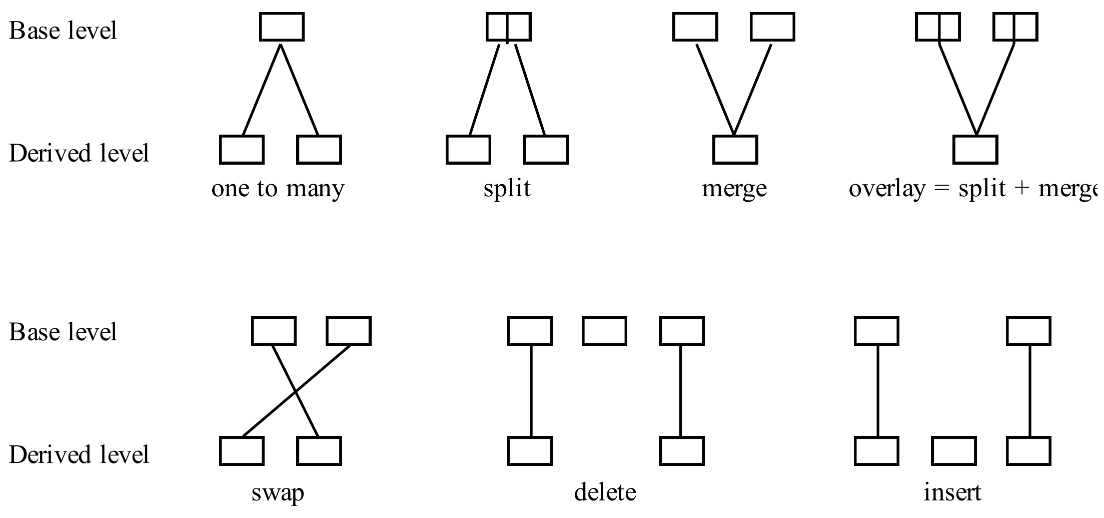

Figure 3. Different operations leading from the base level to some derived level.

6.5 Orthographic Transcription Coding Module

1. Module name: Basic orthographic transcription.

2. Module purpose: Provides basic orthographic tokenisation, event timeline and raw data references. The events are those of TEI. All events and tokens must be within the TEI <u> element. The utterances must supply a timeline reference either in terms of a symbolic sequencing or in terms of a millisecond indication. The words (tokens) should be marked with the <t> element.

3. Coding level: Orthographic transcription.

4. Data sources: Spoken dialogue corpora.

5. Module references: Speech files.

6. Markup declaration:

```
ELEMENT div
ATTRIBUTES
    type: ENUM ( pre | task | mid | post )
    n: NUMBER

ELEMENT u
ATTRIBUTES
    trans: ENUM ( smooth | latching | overlap | pause )
    who: HREF[Resource, person]
CONTAINS w, pause, kinesic, vocal

ELEMENT w

ELEMENT pause
ATTRIBUTES
    dur: TIME

ELEMENT kinesic
ATTRIBUTES
    desc: TEXT

ELEMENT vocal
ATTRIBUTES
    desc: TEXT
```

7. Description:

`Division:` marks overall discourse segments within a dialogue. The divisions are defined relative to the dialogue model structure. Its attributes are:

type The type attribute could be constrained to:

pre Initial greetings and user modeling ("Do you know the system?").

task Solving the task (always a reservation)

mid Transition from one task to another ("Do you want more?").

post Final "Do you want more?" and farewell.

n Divisions are numbered consecutively throughout the whole transcription for each session.

id Unique id.

u: describes what is said, who said it, and what is talked about. Its attributes are:

id The identification of the turn. The identification is unique globally over all the transcriptions of all subjects.

trans The overlap with or transition from the previous utterance.

who The speaker of the utterance. Either S for system or U for user (alias the subject).

Comments: For the system, turns may be divided into several (sub-) utterances each denoted with a separate <u> element. This is feasible for the highly structured phrases typical for a system. In general it may not be possible to separate the turns into utterances with a single, distinct topic, and a more sophisticated markup may become necessary.

Pauses, vocal sounds like 'eh' and non-vocal events (kinesic) like keyclicks are recorded and marked. The attributes have the following restrictions in their use:

pause The value of dur may be '.', '..', and '...' denoting estimated pauses of a duration up to about one second, or a number denoting a measured pause in whole seconds.

kinesic The desc is always key-clicks. These were used as background noise in slight pauses after longer phrases.

vocal The desc is e.g. "øh" and "eh".

Intonation ('?' '!' '.' '-'): The signs '?', '!', and '.' have their usual, orthographic meanings and should be used consistently throughout the transcriptions, but no formal markup is in general defined for these.

8. Example:

<div n="1" type="pre" id="C2-1">

The first division (n="1", -1) of subject two (C2) is a "pre" segment (pre), i.e. initial greetings and user modeling.

<u id="S2-3a" who="S"> <pause id="S2-3a-p1" dur=1><w id="S2-3a-1"> kundenummer</w> - <w id="S2-3a-2">4</w></u>

This is the system's (S) first part (a) of the third turn (3) during conversation with the second subject (2).

<pause id="p1" dur=4>

<kinesic id="k1" desc="key clicks">

<vocal id="v1" desc="øh">

<u id="S3-50a" who="S"> <w id="S3-50a-1">Udrejse</w>! <w id="S3-50a-2">klokken</w> - <w id="S3-50a-3">11</w> - <w id="S3-50a-4">15</w></u>

```
<u id="S3-50b" who="S"><w id=”S3-50b-1”>Er</w> <w id=”S3-50b-2”>det</w> <w id=”S3-50b-3”>rigtigt</w>? </u>
```

Here the '!' after 'Udrejse' is a bit unusual orthograpically, but corresponds to the intonation. After '15' there should have been a '.'. Note how '-' indicates concatenation of prerecorded phrases.

9. Coding procedure:

It is important that the transcription coding is performed and checked very carefully. Since other codings will refer to the numbers in this file, one should not change their identifications.

1. Let coder 1 manually write down the words in orthographic form, marking tokens and who says what, producing “coding1-tmp.sgm”.

2. Let coder 2 manually write down the words in orthographic form, marking tokens and who says what, producing “coding2-tmp.sgm”.

3. Run “compare.prl” and let coder 1 or 2 produce the “coding.sgm”.

4. Let coder 1 or 2 segment into utterances.

5. Run “pause” program to insert pauses semi-automatically (coder 1 or 2).

Note: This is the standard which has an acceptable reliability if the coder in 3-4-5 is suitably trained.

10. Creation notes:

Author: Hans Dybkjær.

Version: 2 (1998/11/22) [version 1 (1996/02/14)].

Comment: First created for use in the Danish Dialogue Project.

Literature: [Dybkjær et al. 1996].

Figure 4. Coding module for orthographic transcription. Note that all elements also provide to the user the standard elements:

Id : ID
TimeStart : TIME
TimeEnd: TIME

7. Raw Data

This list of possible raw data includes, at least:

1. *Sound*: The sound file(s). The sound may be present in two different formats: a) as one big file, or b) split into separate files for e.g. each utterance. Note that in case of overlapping utterances the sound files may contain overlapping speech as well, so that the same sound is part of several sound files.
2. *Video*: Video file(s).
3. *Image*: Picture file(s), e.g. of materials or objects relating to the dialogue.
4. *Data*: Data files containing, e.g., questionnaires or interview questions and their results.
5. *Log*: Log files from software runs.
6. *Notes*: Observer notes.
7. *Transcribed*: Pre-existing transcriptions for which the sound files are no longer available.

Reference to the raw data in the data files is provided by the resource file. In addition to listing the above raw data, the resource file header will also contain information at the meta-level, including:

- Domain, such as train travel; task, such as ordering pizzas; application, such as commercial telephone directory service.
- Human-human (i.e. two or more humans) / simulated human-machine (two humans, one of whom simulates a machine) / human-machine (one human, one machine) / human-human-machine (two or more humans, one or more machines).
- Dialogue participants' numbers and roles (including the role of the machine).
- Speaker characteristics: how speakers were sampled, age and gender of each speaker, speakers' native language, their geographical provenance, accents, social class, their drinking and smoking habits, whether speakers are known to each other, whether speakers are practiced in the dialogue activity [Leech et al. 1998].
- Channel characteristics, including recording characteristics and recording circumstances, involvement of equipment such as telephones, video conferencing systems etc.
- Other environmental characteristics that help explain any peculiarities of the recorded data.
- Who recorded; where; when (date/time).

Higher-level coding files will normally refer to raw data via the transcription file to which they hyperlink. Raw data files may be sound files as well as non-linguistic files, such as screen output or video. In addition, the inventory of data files may include files containing, e.g., questionnaires or interviews, log files from software runs, observer notes etc.

Sound may be represented either in one file or there may be one sound file per utterance. The latter may seem preferable as it would correspond to the division chosen for the basic transcription file and would thus facilitate the reference from the textual representation to the sound representation. Splitting speech files into token-sized files runs the risk of information loss. For instance, pauses may be lost. To a lesser extent, this risk is also present when splitting the speech file into single utterance files.

It may be considered how much time it takes to search one big sound file given a timeline in the transcription. If this can be done in close-to real time no time will be spent on cutting sound files into sub-files.

Top-quality speech data files are produced using one microphone per speaker. This is not always the way speech data files are produced, however, and is not essential to the production of transparent mappings from transcriptions to, e.g., overlapping speech in the data file.

MATE will not enforce any specific format onto non-speech data files, but suggests that video/audio follows current standards, and enforces the possibility to refer to video and audio by means of time in milliseconds.

8. Implications for the MATE Workbench

Some of the implications of the above discussion for the MATE Workbench specification are:

1. The workbench must be able to parse a coding file, using the DTD derived from the coding module referred to in that coding file.
2. The workbench should distinguish two sets of data: the header and the body.
3. The workbench should be able to handle raw data, in particular sound files.

4. The workbench should be able to present the transcription as a time-sequential list or as a musical score.
5. The workbench should be able to present other coding files along with the transcription. One or more coding files may be presented together, either overlaid or parallel to the transcription.
6. When a coding file is presented, its input coding files according to the input coding modules are presented (cf. the previous point).
7. The workbench should support new codings. These should be presented to the user as “coding into” the transcription or other coding files, but in practice a new coding should produce a new coding file.
8. The workbench should support the addition of new coding modules.

9. Representation in XML; Relation to TEI

MATE will use XML for the internal representation of codings. The pros and cons of using XML instead of SGML seem to be that:

- XML is easier for programs to parse, hence speeds up processing.
- XML is easier for humans to read because of less markup optimisation. However, it is the intention that users of the workbench should not need to look at XML files because the XML encoding is hidden behind the interface.
- More free software is available (there is a wider range of Java XML parsers available than there are free SGML Java parsers).
- XML is very likely to become a standard language accessible from a web browser as HTML is today. It should be noted, however, that even if this happens, it will not immediately be possible to display MATE files in a nice way without having a kind of pretty print transformation. This is because the MATE XML tags are customised, not necessarily known by web browsers, and made with no thought on rendering.
- XML is more verbose - i.e. bulkier files - because XML requires all end tags to be present and all attribute values must be quoted. Since most annotation will be constructed by means of XML editors or specialised annotation tools, this is OK. XML files can always be compressed easily and efficiently as they are plain ascii files.
- XML is less powerful - i.e. only simpler contents are allowed. Mixed-content models can only be starred lists of alternatives e.g. (#PCDATA | para | list | font)*. This limitation is really a blessing, as more general mixed content models in SGML are not very useful and are error prone.
- XML has no include or exclude lists. Again this is not terribly useful.
- XML has no unordered lists of children, only ordered ones, i.e. (a , b , c) = a and b and c in that order is ALLOWED, (a & b & c) = a and b and c in any order is NOT ALLOWED. This does not seem to constitute a problem as the layer of the children of the items of a particular other layer can be described in a different file.
- XML has no standard catalogue mechanism yet. SGML or at least standard SGML parsers (e.g. SP) allow one to specify a catalogue which maps SGML external entities to file names. This is helpful when making portable corpora. This is a reasonably serious problem, but at least some XML parsers provide catalogues and it would not be difficult (given an XML API) to write one’s own.
- By not using TEI as the underlying representation MATE is forced to provide import and export functions from and to TEI format. When restricting import to something that has

been exported from MATE, or could have been so (call it “MATE conformant”), there is no problem with import. Given the modular nature of MATE codings it should not be too difficult to export MATE codings into TEI.

Assuming the last point will be solved, XML is certainly more attractive to work with than is SGML and hence TEI.

Notes on exporting to TEI

When exporting the example of Figure 2 into TEI, part of the translation would include:

```
<!-- These entities are intended to be a TEI conformant extension        -->
<!-- as specified in the Guidelines in chapter “Modifying the TEI DTD”    -->
<!-- by the Text Encoding Initiative ((c) ACH, ACL, ALLC).                -->
<!-- This entity file is created for temporary use by the MATE tool.      -->
    <!-- The following elements are deleted   -->
    <!-- The following elements are renamed  -->
    <!-- The following classes are extended   -->
<!ENTITY % n.type “type” >
    <!-- The following elements are revised   -->
<!ENTITY % body ‘IGNORE’ >
```

Moreover, for “top level” elements, such as type, the MATE tool will generate the element DTD from the markup declaration in point 6 (of Figure 2), using a translation like:

```
<!ENTITY % type ‘include’ >
<![ %type; [
<!ELEMENT %n.type; - -      (#PCDATA)                >
<!ATTLIST %n.type;          %a.global;
              type          (month | name | cardinal | ordinal)       #IMPLIED
              value    CDATA          #INHERITED
              TEIformCDATA          ‘type’
>]]>
```

and put it into a file temp.dtd followed by the automatically created part of the DTD:

```
<!ENTITY % newbody ‘INCLUDE’ >
<![ %newbody; [
<!ELEMENT %n.body;          O  ((%n.type;)+) >
<!ATTLIST   %n.body;        %a.global;
%a.declaring;
TEIform       CDATA          ‘body’>
]]>
```

where the list of alternatives (in the example only containing n.type) will contain all the top level elements. This is the TEI-conformant way in which to extend TEI with new coding packages.

It is very clear that no user should be exposed to this kind of markup language specification, let alone be forced to make such specifications by hand.

10. Conclusion

In this document a framework for the definition and representation of markup in spoken dialogue corpora has been specified. The focus has been on the general issues involved in addressing standardisation at several levels of markup, including cross-level markup. Special consideration is given to the usability in terms of ease of use for coders.

The MATE markup framework is based on coding modules which extend and formalise the concept of a coding scheme. A coding module specifies the markup and its intended meaning, as well as the recommended coding process. A coding conforming to this specification is called a coding file of that module. A module is parameterised by other coding modules, meaning that its markup may refer to markup elements of these other modules. Ultimately, all modules refer to markup of one or more transcriptions (e.g. phonetic or orthographic) which in turn are anchored to the raw data via a single resource file. A common timeline is used to align markup of codings that may otherwise be independent. The markup is defined in terms of elements and attributes. The attributes are typed, including ID, TIME, TEXT, and HREF. The formats of coding modules and codings are specified in XML.

Some issues have not yet been fully addressed and will have to be further investigated and developed in WP2 and/or WP3. They include:

- The generation of a DTD for coding files on the basis of a coding module. This should be done automatically by the workbench, so that the user needs not bother with it. How precisely to do this will be further investigated in WP3.
- Phonetic transcription. This will be dealt with in WP2 in which the MATE approach to phonetic transcription will be developed in connection with work on a standard for prosodic annotation.
- Contents of headers of coding files, including transcription files and resource files. The current report already gave some examples of the minimum information which should be included in the different headers. WP2 will elaborate on this and provide a full list. Attention will be paid to the special needs of the resource file and transcription files.
- A more thorough analysis of the timeline and of resource references. The ideas of timeline and resource references will be tested along with the development and test of level markup in WP2 and elaborated as needed.
- An analysis of how to import TEI files and how to export to TEI. Import/export are important functionalities of the workbench and will be developed in this context in WP3.

Other issues may arise during the work on level markup and the workbench. For instance, it may turn out that a monotyped list is a useful attribute type. Such issues will be addressed as they appear. They may add more detail to the framework described in this report, but are not expected to give rise to major revisions.

Acknowledgements: We would like to thank Mimo Caenepeel, Ulrich Heid, Henrik Kasch, Marion Klein, Morten Baun Møller, Claudia Soria, and the MATE participants in general for many valuable comments on earlier versions of this document.

References

[Bernsen et al. 1998] Bernsen, N. O., Dybkjær, H. and Dybkjær, L.: Designing Interactive Speech Systems. From First Ideas to User Testing. Springer Verlag 1998.

[Dybkjær et al. 1996] Dybkjær, L., Bernsen, N.O. and Dybkjær, H.: Evaluation of Spoken Dialogues. User Test with a Simulated Speech Recogniser. Report 9b from the Danish Project in Spoken Language Dialogue Systems. Roskilde University, February 1996.

[Gibbon et al. 1997] Gibbon, D., Moore, R. and Winski, R. (Eds.): Handbook of standards and resources for spoken language systems. Mouton de Gruyter, Berlin, New York, 1997.

[Ide 1996] Ide, N. (Ed.): CES. The Corpus Encoding Standard. http://www.cs.vassar.edu/~ide/CES/CES1.html

[Isard et al. 1998] 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.: Specification of Coding Workbench. MATE Deliverable D3.1, August 1998.

[Klein et al. 1998] Klein, M., Bernsen, N. O., Davies, S., Dybkjær, L., Garrido, J., Kasch, H., Mengel, A., Pirelli, V., Poesio, M., Quazza, S. and Soria, C.: Supported Coding Schemes. Telematics/Language Engineering Project MATE Deliverable D1.1., July 1998.

[Leech et al. 1998] Leech, G., Weisser, M. and Wilson, A.: Draft chapter: Survey and guidelines for the representation and annotation of dialogue. LE-EAGLES-WP4-3.1, Integrated Resources Working Group, 17 April 1998.

[Sperberg-McQueen and Burnard 1994] Sperberg-McQueen, C.M. and Burnard, L. (Eds.): Guidelines for Electronic Text Encoding and Interchange. TEI P3. Text Encoding Initiative. ACH, ACL, ALLC, Chicago, Oxford, April 1994.

Web references:

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

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

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

Annex 1. DTD for Coding Modules

```
<!-- DTD for Mate project Coding modules
     based on D1.2 15 November 98
     NB XML format DTD
-->

<!-- What can appear in textual descriptions -->
<!ENTITY % Text "#PCDATA">

<!-- top level element -->

<!ELEMENT coding-module  (name,desc,coding-level,data-source,uses,
                          declaration,elem-desc,example,
                          coding-proc)>
<!ATTLIST coding-module
          id     ID      #REQUIRED >

<!-- name: name of coding module -->
<!ELEMENT name (%Text;)>

<!-- A description of the types of dialogues this coding module
     applies to -->
<!ELEMENT desc (%Text;) >

<!-- coding-level: Kind of annotation this coding module defines -->
<!ELEMENT coding-level (%Text;)>

<!-- Description of the kind of data the coding module applies to -->
<!ELEMENT data-source (%Text;)>

<!-- A list of coding-modules and raw-data formats that this
     coding-module uses -->
<!ELEMENT uses (module|data)* >

<!-- Hyperlinks to other coding-modules -->
<!ELEMENT module (%Text;)>
<!ATTLIST module
          href      CDATA    #IMPLIED
          xml:link  CDATA    #FIXED "simple"
          show      CDATA    #FIXED "embed"
          actuate   CDATA    #FIXED "auto">

<!-- Hyperlinks to other data formats -->
<!ELEMENT data   (%Text;)>
<!ATTLIST data
          href      CDATA    #IMPLIED
          xml:link  CDATA    #FIXED "simple"
          show      CDATA    #FIXED "embed"
          actuate   CDATA    #FIXED "auto">

<!-- Declaration of the markup elements and their attributes -->
<!ELEMENT declaration (element)*>

<!ELEMENT element (attribute)*>
<!ATTLIST element
          name      CDATA    #IMPLIED
          contains  CDATA    #IMPLIED >

<!ELEMENT attribute (%Text;)>
<!ATTLIST attribute
          name      CDATA    #IMPLIED
          type      CDATA    #IMPLIED >

<!-- A supplementing informal description of the elements and their
     attributes, including purpose, semantics and example -->
<!ELEMENT elem-desc (purpose, semantics, elem-example)*>

<!-- Text description of the purpose of the element, its attributes
     and their values -->
<!ELEMENT purpose (%Text;)>

<!-- Text description of the semantics of the element and its
     attribute values -->
<!ELEMENT semantics (%Text;) >

<!-- An example of the element and its attributes -->
<!ELEMENT elem-example (%Text;) >

<!-- An example of the use of this coding-module -->
<!ELEMENT example (%Text;) >
```

```
<!-- Coding procedure for this coding-module -->
<!ELEMENT coding-proc (%Text;)>
<!ATTLIST coding-proc
          name          ID        #REQUIRED >
```