Project ref. no.	IST-2000-26095
Project title	NITE: Natural Interactivity Tools Engineering

Deliverable status	Restricted
Contractual date of delivery	1 February 2002
Actual date of delivery	November 2002
Deliverable number	D2.2
Deliverable title	The NITE Markup Framework
Type	Report
Status & version	Final
Number of pages	41
WP contributing to the deliverable	WP2
WP / Task responsible	NISLab
Author(s)	Laila Dybkjær, Niels Ole Bernsen, Jean Carletta, Stefan Evert, Mykola Kolodnytsky, and Tim O'Donnell
EC Project Officer	Philippe Gelin
Keywords	Markup framework, raw data, coding module, coding file, meta-data, natural interactivity, multi-level, cross-level, cross-modality
Abstract (for dissemination)	This deliverable discusses, and proposes solutions to, the general issues involved in creating a general standard for the markup of natural interactivity corpora. It presents the NITE markup framework including raw data, coding modules and coding schemes, coding files, and meta-data, and it describes important issues to be addressed. In addition, the deliverable presents the proposed realisation of the markup framework in two of the three software strands in the NITE project, i.e. the NITE WorkBench NWB and the NITE XML Toolkit, NXT, respectively.

NITE Deliverable D2.2

The NITE Markup Framework

November 2002
Final

Authors

Laila Dybkjær[1], Niels Ole Bernsen[1], Jean Carletta[2], Stefan Evert[3],
Mykola Kolodnytsky[1], and Tim O'Donnell[2]

1: NISLab, University of Southern Denmark, Denmark. 2: HCRC, Edinburgh, UK. 3: IMS, Stuttgart University, Germany.

Section responsibilities

Sections 1-7: NISLab

Section 8 and Appendices A and B: HCRC and IMS

Contents

1 Introduction

This document presents a first version of the NITE markup framework. The NITE (Natural Interactivity Tools Engineering, see nite.nis.sdu.dk) markup framework builds on the MATE markup framework described in [Dybkjær et al. 1998]. However, whereas the MATE (Multi-level Annotation Tools Engineering) project only addressed spoken language dialogue corpus annotation, NITE addresses the general case of natural interactivity corpus annotation.

A natural interactivity corpus is a body of natural interactivity data which has been recorded. Natural interactivity data includes human-human and human-machine communicative behaviour involving some or all of the following modalities: speech, gesture, facial expression, gaze, head and body posture, object manipulation as part of the communication, etc. The corpus typically consists of audio files and/or video files. We shall refer to such data which has not yet been marked up in any way, as raw data.

A natural interactivity corpus can be an almost inexhaustible source of information which can be used for many different purposes. To turn a raw data corpus into an immediately useful information source, the information which is relevant for the purpose in question must be reliably marked up using some coding scheme or other, for audio data typically starting with some form of transcription. 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 extract and exploit the marked-up information for research, industrial, or other purposes. If the markup follows common standards, its re-use becomes much easier. Moreover, it then becomes possible to develop general tools which support markup using coding schemes which follow those common standards.

However, so far there are no firmly established standards for annotation of natural interactivity data. At best, emerging standards are appearing in a few limited sub-areas of natural interactivity. For instance, MPEG-4 (http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm) may be viewed as a de facto standard for basic facial markup. However, most current natural interactivity coding schemes are home-grown, tailored to the particular needs of their creators, and not widely used by the community at large.

Several initiatives are currently underway collecting information on best practice and standards in the field and/or aiming to produce recommendations and propose standards. Among these initiatives are ISLE (International Standards for Language Engineering) which includes a working group on natural interactivity and multimodality (ISLE NIMM, isle.nis.sdu.dk), OLAC (Open Language Archives Community, www.language-archives.org/docs.html), and the ISO TC37/SC4 group (atoll.inria.fr/RLT/TC37SC4General.ppt). In addition to producing surveys of existing natural interactivity and multimodality data resources, coding schemes, and coding tools, the ISLE NIMM working group aims to produce guidelines for the creation of NIMM data resources and annotation schemes. OLAC aims to create a worldwide virtual library of language resources, rather than natural interactivity data resources more generally, by developing consensus on best current practice for the digital archiving of language resources and by developing a network of interoperating repositories and services for housing and accessing such resources. The scope of ISO TC37/SC4 is to specify a platform for designing and implementing linguistic resource formats and processes, which means that this group only addresses a limited aspect of natural interactivity. Further references to related bodies and activities can be found in the ISLE deliverable D8.2 (isle.nis.sdu.dk). The present report discusses, and proposes solutions to, the general issues involved in creating a general standard for the markup of natural interactivity corpora. Section 2 introduces the NITE markup framework and describes the problems to be addressed. Sections 3 through 5 discuss raw data (Section 3), coding modules and coding schemes (Section 4), and coding files (Section 5), respectively. In addition, each of these sections discusses related meta-data requirements. Section 6 discusses some challenging issues for the markup framework, such as long-range dependencies and cross-level markup. Sections 7 and 8 present the proposed realisation of the markup framework in two of the software strands in the NITE project, i.e. the NITE WorkBench NWB (Section 7) and the NITE XML Toolkit, NXT (Section 8), respectively.

The NITE consortium follows a three-tiered approach in its development of NITE Markup Framework-conformant coding tools for natural interactivity data annotation. Based on a common requirements specification, the commercial partner Noldus Information Technology works towards meeting some of the core NITE requirements in future releases of the company's widely used The Observer software package (see www.noldus.com). Based on the same specification, the NITE academic partners are developing two different open source versions of the NITE toolset, i.e. the NITE WorkBench for Windows, or NWB, which implements the NITE specifications in a Windows environment familiar to all or most users, and the NITE XML Toolkit, or NXT, which is platform independent and requires XML skills of its users. For more details on The Observer, NWB and NXT, see nite.nis.sdu.dk, as well as, for NWB and NXT, Chapters 7 and 8 below, respectively.

Based on the markup framework solution proposed below, the framework will be implemented, tested, and further refined in the NITE software, in particular in NWB and NXT. A broad selection of coding schemes for markup of natural interactive communicative behaviours are presented in NITE deliverable D2.1 [Serenari et al. 2002]. The issue of state-of-the-art natural interactivity coding scheme selection will be addressed again in a slightly different context in the forthcoming NITE deliverable D2.3. In this deliverable, selected best practice coding schemes are discussed for possible inclusion in the NITE software and example schemes are represented in terms of coding modules in accordance with the NITE Markup Framework.

2 The NITE markup framework

The primary objective of NITE is to build an integrated best practice workbench for multi-level, cross-level, and cross-modality annotation, retrieval, and exploitation of multi-party natural interactive human-human and human-machine dialogue data. The NITE software will implement and support use of the NITE markup framework. The NITE markup framework will build on, and extend, the MATE standard markup framework to accommodate the needs of current and emerging coding modules for natural interactivity coding. The notion of a coding module was introduced in the MATE markup framework as an elaboration of the notion of a coding scheme and will be discussed at length in Section 4.

The need for annotating and exploiting natural interactivity data is increasing rapidly. This is, first and foremost, due to the emergence of advanced systems which enable natural interaction with their users beyond what has been possible for more than a decade in the case of, e.g., spoken language dialogue systems. The development of such systems requires annotated data and is expected to be strongly facilitated by coding scheme standardisation and new, more powerful coding tools. However, the range of purposes for which coded natural interactivity data may be used is in principle unlimited. Some examples are the use of annotated data for training a particular system component and for investigating a new class of natural interactive behaviours and their coordination with behaviours which are well described in the research literature already.

The scope of a natural interactivity markup framework must be broad enough to support the multitude of annotation purposes which people actually have, whether these purposes are academic or industrial, purely theoretical or purely developmental, or any combination of those. Furthermore, the markup framework should include a description of coding modules, a description of raw data and coding files, a description of how several codings of the same raw data can co-exist and be interrelated (in cross-level and cross-modality markup), and a description of how to handle other important issues, such as time alignment, synchronous and asynchronous phenomena, long-range dependencies, relationships, overlaps, quality of coding, markup semantics, and annotation of multi-party natural interactive conversation.

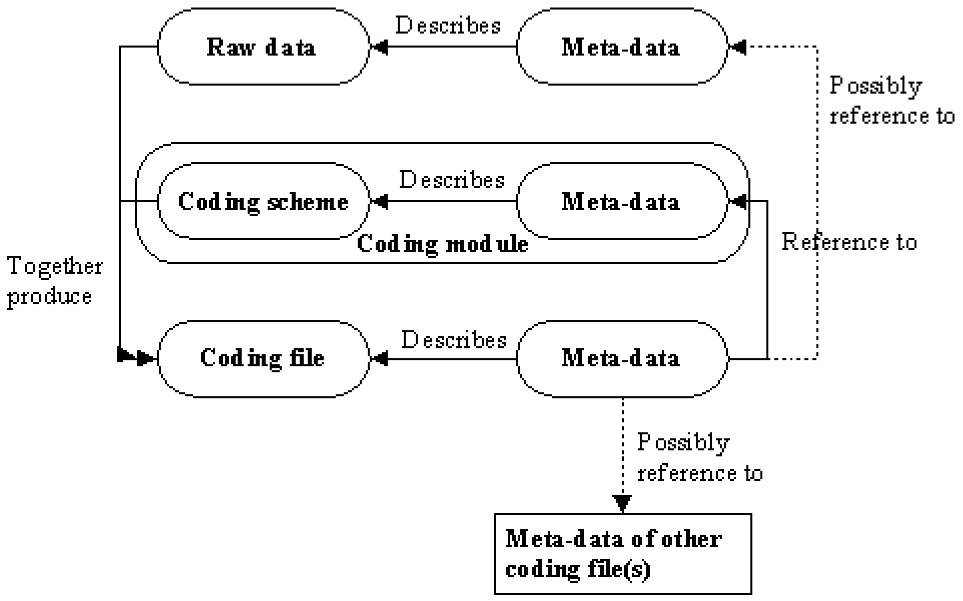


Figure 2.1. General NITE markup framework file structure.

The NITE markup framework file structure is shown in Figure 2.1. The basic annotation process requires a coding module and either raw data (audio and/or video) or an already annotated coding file referencing raw data. A coding module consists of a coding scheme and meta-data information. When applied to raw data or to an existing coding file, the coding scheme produces a new coding file.

Coding file meta-data references the meta-data of the coding module. Coding file meta-data may also directly reference the meta-data of the raw data. Otherwise, raw data reference is made indirectly via reference to the meta-data of another coding file. Links are also made to the meta-data of other coding files if these are referenced in the coding file.

3 Raw data and its meta-data

By raw data we understand data files which should not be edited because they constitute the basic information source. Any added information (value) should be inserted via references to the raw data and not in the raw data files themselves.

Raw data files will often be dynamic signal files such as audio or video files but raw data files may also be, e.g., static image files or pure text files as appears from the following list of possible raw data files considered in NITE:

1. *Audio file(s):* The sound may be present in two different ways: a) as a single large 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 may be part of several sound files. A common representation format for sound files is wav but there are others as well.
2. *Video file(s):* The video file may present a single window showing the scene from one particular perspective or it may present several windows showing the same scene from different angles and from different distances. Common representation formats for video files are, e.g., avi and mpeg.
3. *Image*: Static picture file(s) showing, e.g., face stills or materials or objects relating to the dynamic interaction. Images may come in may different formats, e.g. gif, jpeg, and tiff.

5. *Log*: Log files from software runs. These are text files.

4. *Data*: Data files containing, e.g., questionnaires or interview questions and their answers. Such data files are often text files.

6. *Notes*: Observer notes. These are usually in text format.
7. *Transcribed*: Pre-existing transcriptions for which the sound files are no longer available. These are typically text files, e.g. represented in XML format.

Any raw data file should be accompanied by a meta-data file which describes the raw data file and references it. A meta-data file for raw data may include many types of information, depending on the type of raw data it describes. The following list provides examples of information entries which have been found useful in meta-data files.

1. raw data referenced
2. date of creation of raw data
3. date of creation of meta-data
4. name(s) of creator(s) of raw data
5. name(s) of creator(s) of meta-data
6. location for creation of raw data
7. purpose of creation of raw data, modalities involved, etc.
8. interacting participants, numbers and roles (including the role of the computer, if any)
9. speaker characteristics, including 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 the speakers are known to each other, whether the speakers are practiced in the dialogue activity performed
10. size of raw data (measured in duration, number of dialogues or Mb)
11. accessibility (fee or free, web site, contact information)
12. file technicalities, including file format, compression information, etc.
13. recording setup description, including environment, equipment, microphone and camera placements, etc.
14. application description, including subjects, task, scenario, instructions, nationality, language, native or non-native speakers, domain knowledge, etc.
15. references to other raw data which are important to the raw data described and to annotations of the raw data described
16. references to, e.g., literature, reports and publications, etc., providing more details on the raw data or some aspect thereof

17. notes

There is no such thing as an exhaustive general-purpose list of meta-data information on raw data, so the above list merely presents potentially useful examples of meta-data information. It should always be possible to add new information entries to the raw data meta-data file. However, recommendations on the information which should be included, as a minimum, in raw data meta-data files will shortly be available in ISLE deliverable D8.2 which provides guidelines for the creation of natural interactivity and multimodal data resources. This ISLE report focuses on which information to include in the documentation when creating audio, video or image files. The report also includes recommendations on how to make good quality audio/video recordings. An example is included of a raw data video file and its meta-data description according to the recommendations made.

NITE does not recommend any particular raw data file formats but suggests that video/audio files follow current standards. It is recommended to enable reference to video and audio data in terms of time in milliseconds.

4 Coding module and coding scheme

A *coding module* includes or describes everything that is needed in order to perform a certain kind of markup of a particular natural interactivity corpus. It describes itself and prescribes what constitutes a coding, including the coding scheme with its tag set, and the relations to other codings. In The Observer, the configuration file in some sense corresponds to a coding module but it should be noted that a configuration file is not identical to a coding module as described in the following.

We define a *coding scheme* as a set of concepts of phenomena to be found in corpora. Typically, a coding scheme will also include a syntactically defined set of tags for marking up those phenomena. A coding scheme is part of a coding module.

A valid coding must follow the prescriptions of the selected coding module (or configuration file) and use some subset of the tags included in the coding scheme.

The core part of a coding module is the set of phenomena we are interested in marking up when using this particular coding module, and which is also referred to as the coding scheme. However, a coding scheme-cum-tag set is not very useful without accompanying information about purpose, scope, semantics, intended use, etc. This additional information may be regarded as meta-data serving to inform a user on the coding scheme and its use. From a usability point of view, this additional information is just as important as the coding scheme itself. Without the information, the coding scheme is unlikely to be useful to anyone except perhaps its creator at creation time.

A coding module includes two types of information. One type is mainly intended for the user and another type is mainly intended for the system. The information intended for the user may be viewed as the coding scheme concepts together with their meta-data information. The information intended meant for the system basically corresponds to a tagset.

The name of the coding module is intended for both the user and the system.

Information mainly intended for the user includes:

1. author(s)
2. version
3. notes (references to literature, validation information, comments, etc.)
4. purpose of the coding module
5. coding level(s) (e.g. dialogue acts, hand gesture, nose wrinkles, …)
6. description of data source type(s) required for use of the coding module
7. explanation of references to other coding modules
8. coding procedure
9. coding example showing the coding scheme markup in use
10. clear description of each phenomenon, example(s) of each phenomenon. Phenomena such as gestures and facial expressions may be illustrated in static or dynamic images to further help the user identify their occurrence in the data (cf. Figure 4.2)

Information primarily intended for the system includes:

11. a markup declaration, possibly hierarchically ordered, of the tags for the (individually named) phenomena which can be marked up using the coding module
12. coding files referenced

In this report, we shall not go into more detail concerning actual recommendations for coding module and coding scheme creation. The ISLE deliverable D9.2 will shortly provide guidelines for the creation of natural interactivity and multimodal coding schemes and modules. This ISLE report surveys existing proposals for a markup framework in the natural interactivity and multimodal area as well as the information to include in a coding module. An example is included of a coding module and the application of its coding scheme to a natural interactivity and multimodal raw data file.

4.1 Coding module explained

This sub-section briefly explains and illustrates the coding module entries listed in the preceding section. It should be noted that examples are typically *not* related across entries. An example is merely used to illustrate the particular entry under which it is found. A full example of a coding module is presented in Section 4.2.

Name: The name provides a title of the coding module. The name is used for reference.

Example: MyGestures.

Author(s): The name of the person(s) who created the coding module.

Example: Laila Dybkjær.

Version: The version number of the coding module. A coding module may get updated and it is important to clearly indicate its current version number.

Example: Version 1.0

Notes: Notes may contain any kind of useful description or reference, such as references to literature which describes the coding module and/or its use, or a description of, or reference to, a validation of the coding module.

Example: The coding module has been validated by NN. Results can be found in [ref.].

Purpose of the coding module: Typically, this entry describes the purpose for which the coding module was first created.

Example: Coding of hand gesture and emotions in Asterix cartoons.

Coding level(s): Description of the coding level(s) covered by the coding module.

Example: Head posture, gaze, dialogue acts.

Description of data source type(s) required for use of the coding module: Description of the data file type pre-conditions for using the coding module. For example, a pre-condition may be that an orthographic transcription exists. A corpus (raw data) will always be required although it may often be only indirectly referenced, e.g. via a transcription.

Example: Human-computer dialogues (raw data) and phonetic transcription (coding file).

Explanation of references to other coding modules: Explanation of which other coding modules must be referenced. If the coding module assumes that there are references to other levels of markup then the references should be to the coding module(s) which cover this/these levels.

Example: Phonetic transcription module, facial emotions module.

Coding procedure: Description of how the coding module should be applied to a corpus in order to produce a new coding. The coding procedure is important to ensuring the reliability of the coding and thus to its quality. The coding procedure should include, cf. [Dybkjær et al. 1998]:

1. Description of the coders: their number, roles and required 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).

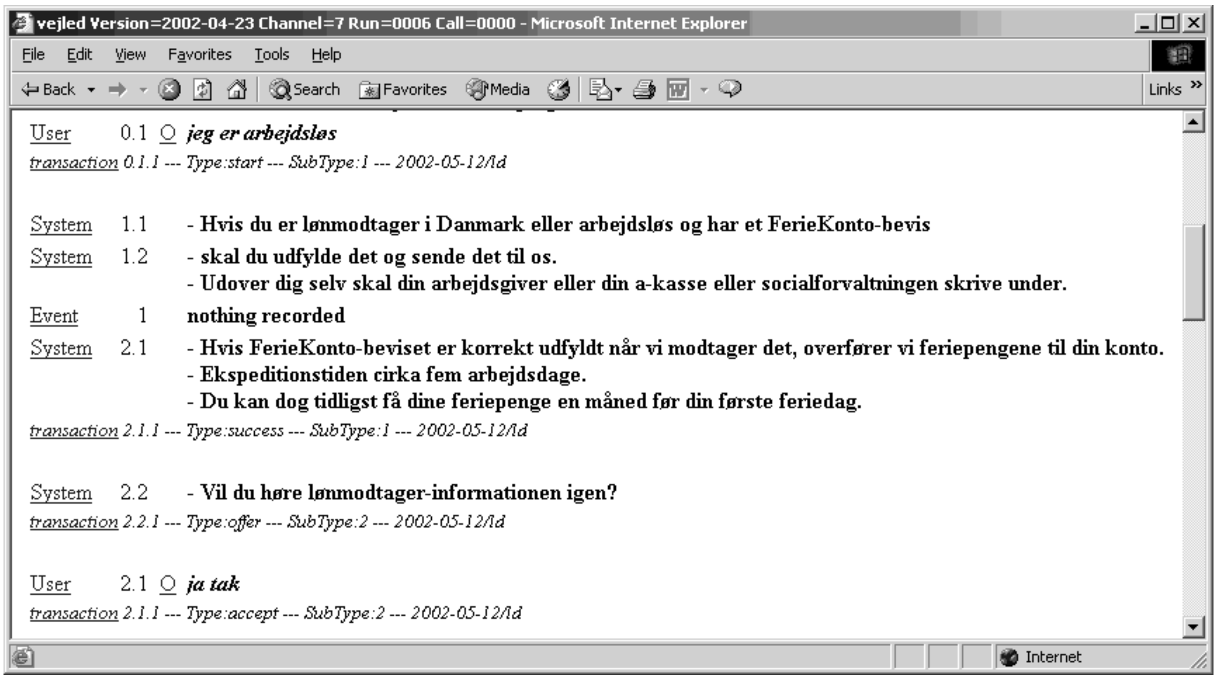
The coders are generically referred to as Coder 1, Coder 2, ... Occurrences of the same coder number indicate that the same coder should be used. Different coder numbers indicate that the coders must be different individuals. In the interest of promoting increased coding reliability, we propose that coders should be able to express uncertainty wrt. to how to tag a particular token in the data.

Example: Coder 1 annotates the corpus. Coder 2 checks the resulting annotation file and makes corrections if necessary. Corrections should be saved in a new copy of the annotation file. If the amount of corrections exceeds 10% then let Coder 3 check the corrected annotation file and add any needed corrections.

Note that a list of the actual coders will not exist until the coding module is being applied, at which point the coders are described in the meta-data (header) of the coding file in terms of their role(s) in the coding procedure, their background, training, coding experience, and possibly other properties, cf. Section 5.

Coding example showing the markup in use: This could be a snippet from an actually annotated file or it could be a constructed example. The purpose is to give users of the coding module an idea of what the markup looks like when applied.

Example: See Figure 4.1. The figure shows a dialogue transaction coding module applied to a spoken dialogue.



vejled Version=2002-04-23 Channel=7 Run=0006 Call=0000 - Microsoft Internet Explorer

```
User     0.1  ○ jeg er arbejdsløs
transaction 0.1.1 --- Type:start --- SubType:1 --- 2002-05-12/ld

System   1.1    - Hvis du er lønmodtager i Danmark eller arbejdsløs og har et FerieKonto-bevis
System   1.2    - skal du udfylde det og sende det til os.
                - Udover dig selv skal din arbejdsgiver eller din a-kasse eller socialforvaltningen skrive under.
Event      1    nothing recorded
System   2.1    - Hvis FerieKonto-beviset er korrekt udfyldt når vi modtager det, overfører vi feriepengene til din konto.
                - Ekspeditionstiden cirka fem arbejdsdage.
                - Du kan dog tidligst få dine feriepenge en måned før din første feriedag.
transaction 2.1.1 --- Type:success --- SubType:1 --- 2002-05-12/ld

System   2.2    - Vil du høre lønmodtager-informationen igen?
transaction 2.2.1 --- Type:offer --- SubType:2 --- 2002-05-12/ld

User     2.1  ○ ja tak
transaction 2.1.1 --- Type:accept --- SubType:2 --- 2002-05-12/ld
```

Figure 4.1. Example of an annotation file showing a user-system dialogue over the telephone (in the Danish). The coding module applied allows the coder to mark up different transactions defined in the coding scheme, such as start, success, offer, and accept.

Clear description of each phenomenon, example(s) of each phenomenon: The descriptions provided under this entry are essential to communicating the semantics of the concepts of the coding scheme. It should be explained as clearly as possible how each concept-tag pair should be applied during markup. Any uncertainty left by the descriptions and examples provided will translate into unreliable coding, inter-coder disagreement, etc.

Example: An AU is a minimal visible action. It corresponds to the action of a muscle or a group of related muscles. Each AU describes the direct effect of muscle contraction as well as secondary effects due to movement propagation and the presence of wrinkles and bulges.

AU1 describes an inner brow raise. AU2 describes an outer brow raise. Figure 4.2 illustrates the use of AU1+AU2.

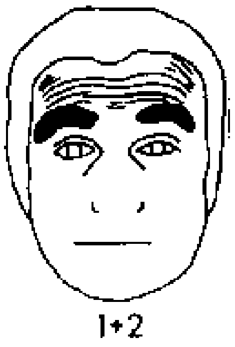


Figure 4.2. Inner and outer brow raise (AU1+AU2).

We stop the example here. The tags were taken from the FACS coding scheme which includes far more tags than explained here, see e.g. [Knudsen et al. 2002].

Information primarily intended for the system includes:

A markup declaration of the tags for the phenomena which can be marked up using the coding module: The tag set declaration can be presented in several different ways. If there is tools support for annotation, the declaration may, e.g., be represented in terms of a dtd (document type definition). A dtd is meant to be read by a computer rather than by humans. So, for many users, such a declaration may cause problems. It may also be that the coding tool supports a more user-friendly way of specifying the markup declaration, so that the user does not have to think about what is a good format for the computer. If not meant to be read by a computer, the markup declaration may be written in a way which makes it easy for humans to read and understand. As the following examples illustrate, the markup declaration may often appear hard to understand if there is no accompanying information. This is exactly the main reason why we recommend that all the entries of the coding module be filled. The additional information is crucial to facilitate reuse by other annotators.

Example1: See entry 12 and Figure 4.2.2 in Section 4.2. This shows the set of markup phenomena for ToonFace (www.media.mit.edu/) which is a coding scheme intended for rapid creation and animation of interactive cartoon faces.

Example2: This example is taken from a SmartKom gesture annotation scheme which has been entered into the Anvil tool (www.dfki.de/~kipp/anvil) using XML. Anvil has no other way of entering a coding scheme apart from the XML-based one.

```
<?xml version='1.0'?>
<annotation-spec>
<head>
<valuetype-def>
<valueset name="phaseType">
  <value-el color="#eeee00">
   prep
   <doc>
    Preparation phase, bringing arm and hand into stroke
    position. Note that changing hand shape before/after moving
    the arm belongs to the preparation, too. Also code position
    info.
   </doc>
  </value-el>
  <value-el color="#dd0000">
   stroke
   <doc>
    The most energetic part of the gesture movement. <b>Encode all
    other attributes for this gesture phrase in the stroke
    element!</b>
   </doc>
  </value-el>
  <value-el color="#aa0044">
   beats
   <doc>
    Only for non-beat gestures! A number of successive strokes
    (beats); all strokes should be covered by this phase. If there
    is a hold in-between then continue with prep.
   </doc>
```

```
    </value-el>
    <value-el color="#ee8800">
     hold
     <doc>
       A phase of stillness just before or just after the stroke,
       usually used to defer the stroke so that it coincides with a
       certain word. <b>When annotating an element as "hold" do not
       annotate any other attributes!</b>
     </doc>
    </value-el>
    <value-el color="#ee00ee">
     recoil
     <doc>
       Directly after the stroke the hand may spring back so as to
       emphasize the harshness of the stroke.
     </doc>
    </value-el>
    <value-el color="#00bb33">
     retract
     <doc>
       Retraction. Movement back to rest position. In sitting
       position this is usually the arm rest, the lap or folded
       arms. <b>Encode no other attributes in a "retract" element.</b>
     </doc>
    </value-el>
    <value-el color="#00aa66">
     partial-retract
     <doc>
       Retraction movement that is stopped midway to open another
       gesture phrase. Code position info here.
     </doc>
    </value-el>
   </valueset>
   <valueset name="gestureType">
    <value-el color="#8cda8e">beat</value-el>
    <value-el>deictic</value-el>
    <value-el>emblem</value-el>
    <value-el>iconic</value-el>
    <value-el>metaphoric</value-el>
    <value-el>adaptor</value-el>
   </valueset>
 </valuetype-def>
 </head>
 <body>
 <!-- <track-spec name="wave" type="waveform" height="1.5" /> -->
 <group name="utterance">
   <track-spec name="smartkom" type="primary">
```

```
      <attribute name="text" valuetype="String" display="true"/>
    </track-spec>
    <track-spec name="user" type="primary">
      <attribute name="text" valuetype="String" display="true"/>
    </track-spec>
  </group>
  <track-spec name="head" type="primary" >
    <attribute name="motion" display="true">
    <value-el>shake</value-el>
    <value-el>nod</value-el>
    </attribute>
  </track-spec>
  <group name="gesture">
    <track-spec name="phase" type="primary">
      <attribute name="type" valuetype="phaseType" display="true">
      <doc>
       My phase description is based on the phases postulated by
       Kendon and McNeill (1992), later on extended by Kita et
       al. (1999).
      </doc>
      </attribute>
    </track-spec>
    <track-spec name="phrase" type="span" ref="gesture.phase">
      <attribute name="category" valuetype="gestureType" display="true" />
    </track-spec>
    </group>
  </body>
  </annotation-spec>
```

Coding files referenced: If there is tools support for the coding module, the tool must be informed about which kinds of coding files should be referenced, i.e. if, e.g., an orthographic transcription or a hand gesture coding file should be referenced, cf. above. When applying the coding module, the user must inform the tool about which actual coding files will be referenced. This/these coding file(s) must match the description of references to other coding modules belonging to the chosen coding module.

> *Example:* If the coding module has a reference to an orthographic transcription module, then the file ort_trans.xml could be a coding file which has been produced by using an orthographic transcription module and which fulfils the reference requirements of the coding module.

4.2 Example of a coding module

1. Name:

Toonface.

2. Author(s):

Kristinn R. Thórisson at the M.I.T.Media Laboratory: http://www.media.mit.edu/

An extension of ToonFace, CharToon, has been developed at CWI under the supervision of P.J.W ten Hagen as part of the European project Facial Analysis and Synthesis of Expressions (FASE), 1997-2000. See H. Noot and M. Ruttkay: CharToon 2.0 manual. INS-R0004, ISSN 1386-3681. 2000. Can be downloaded from: http://www.cwi.nl/ftp/CWIreports/INS/INS-R0004.ps.Z

3. Version:

N/A.

4. Notes:

A description of the ToonFace animation framework:
http://xenia.media.mit.edu/~kris/gandalf.html

References to additional information on the coding scheme:

Thórisson, K. R.: ToonFace: A System for Creating and Animating Cartoon Faces. Learning and Common Sense Section. Technical Report 1-96, 1996. Can be downloaded from: http://xenia.media.mit.edu/~kris/ftp/toonface.pdf

For more details on FASE, see:

The Facial Analysis and Synthesis of Expression homepage: http://www.cwi.nl/projects/FASE/

The Facial Analysis and Synthesis of Expression Chartoon homepage:
http://www.cwi.nl/projects/FASE/CharToon/

5. Purpose of the coding module:

ToonFace is a coding scheme for coding 2D facial expression with limited detail.

6. Coding level(s):

Facial expression.

7. Description of data source type(s) required for use of the coding module:

The application of the coding scheme produces a text file. The application of this file produces an animated face which requires video rendering.

8. Explanation of references to other coding modules:

None. Note the following pre-condition, however. The coding scheme is meant to define a set of parameters for controlling a 2D facial model. A representation of a face with a neutral expression is therefore required as point of departure.

9. Coding procedure:

It is probably sufficient to let one expert coder make the encoding and then let another coder check the result by watching the animated face.

10. Coding example showing the markup in use:

ToonFace consist of two parts, an editor and an animation engine or animator. The editor supports markup via a graphical interface. Faces are constructed in the editor by drawing a number of polygons. Figure 4.2.1 shows examples of faces created using Toonface

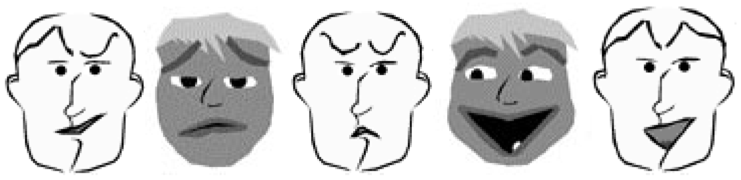


Figure 4.2.1. Examples of faces created with Toonface.

11. Clear description of each phenomenon, example(s) of each phenomenon:

A face is divided into seven main features: Two eyebrows, two eyes, two pupils and a mouth. The eyebrows have three control points each, the eyes and mouth four, and the pupils one each, see also Figure 4.2.2. The ToonFace editor allows manipulation of a face by adjusting control points, cf. entry 12.

12. Markup declaration of the tags for the phenomena which can be marked up using the coding module:

Control points that can be animated are given the codes shown in Figure 4.2.2. These points were selected to maximize the expressiveness/complexity tradeoff. In the case of points that can move in two dimensions, each dimension is denoted as either "h" for horizontal or "v" for vertical.

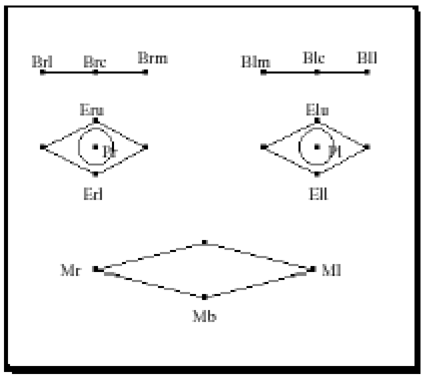


Figure 4.2.2. Codes used for the animated control points.

The following is a complete list of all one-dimensional motors that can be manipulated in a face (control point number in brackets):

Brl = brow/right/lateral [3]; Brc = brow/right/central [2]; Brm = brow/right/medial [1]

Bll = brow/left/lateral [6]; Blc = brow/left/central [5]; Blm = brow/left/medial [4]

Eru = eye/right/upper [7]; Erl = eye/right/lower [9]

Elu = eye/left/upper [8]; Ell = eye/left/lower [10]

Plh = pupil/right/horizontal [15]; Plv = pupil/left/vertical [15]

Prh = pupil/right/horiz [16-h]; Prv = pupil/right/vertical [16-v]

Mlh = mouth/left/horizontal [14-h]; Mlv = mouth/left/vertical [14-v]

Mrh = mouth/right/horizontal [13-h]; Mrv = mouth/right/vertical [13-v]

Mb = mouth/bottom [12]

Hh = head/horizontal [17-h]; Hv = head/vertical [17-v]

Horisontal motion is coded as 0, vertical as 1. Each of the motors can move a control point between a minimum and a maximum position (for a given dimension). Thus, max and min values mark the limits of movement for each motor. For the eyes and head, these are given in degrees, (0,0) being straight out of the screen; upper left quadrant being (pos, pos), lower left quadrant being (pos, neg)

13. Coding files referenced:

N/A.

5 Coding files and their meta-data

The application of a coding module should follow a coding procedure and produce a coding file. A coding file is also sometimes referred to as an annotation, annotation file, or annotated data. A coding file consists of two parts, i.e. meta-data information of relevance to the actual coding and the actual coding itself. The meta-data part is often referred to as the *header* while the actual coding is called the *body.*

The meta-data file for a coding file (the header file) should include at least the following information:

1. reference to the coding file body
2. dates of creation, coding, revisions
3. name(s) of coder(s) and their characteristics
4. version number
5. reference to the applied coding module
6. references to the actual coding files or raw data referred to in the applied coding module
7. names of the (numbered) coders in the coding procedure
8. purpose of creation
9. level(s) and modality(-ies) annotated
10. history of coding procedure
11. notes
12. references to literature, reports, publications, etc.

There have been many discussions on meta-data in general and not least on which information to include when. The above list is based on common sense and experience among the NITE participants. The list is in no way meant to be exhaustive but merely presents prompts for information which we consider crucial for the documentation of coding files in general. More specialised information should of course be added as needed in the particular case. If a tool is available, then it should support the addition of information entries. More information on meta-data can be found in the ISLE report D10.2 [Broeder et al. 2002].

6 Important issues in natural interactivity coding

In the following, we briefly explain a range of important issues, structural and otherwise, in natural interactivity coding, which were identified in our requirements specification of the NITE software, cf. [Carletta et al. 2001].

Cross-level coding. Annotation of a particular modality, such as speech, may be done at different levels, such as orthographic transcription, prosody markup, or morpho-syntactic markup. A level is some level of abstraction at which selected information in the raw data can be identified, described, annotated, and analysed. Cross-level coding consists in representing various types of links (of co-existence, cueing relationships, etc.) between phenomena at different levels of annotation. An example is the markup of prosodic cues to speech acts.

Cross-modality coding. Annotation is often concerned with the coding of a single modality only, such as gesture, speech, or facial expression. However, annotators may also want to link across modalities. This is called cross-modality coding. For example, a user may want to cross-reference emotional cues in prosody and facial expression.

Individual interactions and group interactions. Individual interactions may overlap while group interactions involve at least two parties performing a shared action, e.g. handshaking. Annotation software must support the representation of both kinds of interaction.

Intersecting hierarchies. These are also called overlapping hierarchies and refer to the fact that the phenomena one wants to mark up do not always nest in a nice and regular way but may overlap in different ways. For example, the sentence "John likes Mary" may be tagged: <s/<a/ John <b/ likes /a> Mary /b>/s>, where the entire sentence is tagged as an *s*. The words *John likes* are tagged as an *a*, and the words *likes Mary* as a *b*. Because the *a* and the *b* overlap, the document has no convenient representation in SGML or XML, cf. (http://www.hit.uib.no/claus/goddag.html). Since overlapping hierarchies occur quite frequently we need to find solutions for handling them in the NITE software.

Long-range dependencies. A phenomenon may involve long-range dependency across utterances, turns and speakers, for instance in co-reference or causal relationships. Long-range dependencies impose demands on visualisation which must be taken into account. For example, we might want to display long-range dependencies through horizontal, within-level link lines.

Synchronous and asynchronous phenomena. Phenomena which the user wants to tag may happen at the same time, i.e. synchronously, or at different times, i.e. asynchronously. Annotation software must support the representation and visualisation of both synchronous and asynchronous phenomena and their interrelations.

Time alignment. An annotation tool must enable alignment of raw data and different annotations of the raw data to the same timeline. The purpose of a timeline is to provide a common anchor point in time for transcriptions and other codings, and to sort out the sequentialisation of the markup.

In the following two sections, we briefly describe the realisation of the proposed markup framework by the NWB and the NXT approach, respectively.

7 The NITE WorkBench approach

The NITE WorkBench approach builds on the idea illustrated in Figure 2.1, i.e. that individual coding files and meta-data files are kept separate. Files are structured and interrelated via references. In this way, files may form hierarchies or layers of relationships. This provides a high degree of modularity.

Files belonging together are organised in a project file. A project file contains references to files which are interrelated, such as raw data files which belong together, and to their annotations. The project file must be updated whenever a new file belonging to it has been created and should be referenced.

A project file has a name and consists of:

1. references to one or more raw data files which belong together,
2. references to a meta-data file for each raw data file,
3. references to coding files containing annotations (including transcriptions) of the raw data, if any,
4. references to a meta-data file for each coding file,
5. references to other files belonging to the project, such as query files.

This structure of references can be easily handled by using a relational database.

7.1 Storing annotated data in a database

To satisfy the list of requirements stated in [Bernsen et al. 2002] as well as to be able to solve the important issues in natural interactivity coding mentioned in Section 6, a relational database also appears to be a good choice as the underlying mechanism for data storage in the NITE WorkBench. When data is stored in a relational database one may basically view the data as stored in a number of interrelated tables. One table will contain a timeline, another table may contain an orthographic transcription in which each word is linked to the timeline, a third table may contain dialogue act tags linking to the words in the orthographic transcription and thus indirectly to the timeline. In case of a video with no transcription, there will still be a timeline and, e.g., gesture tags will then link directly to the timeline.

7.2 Entering and storing coding module information

The data structure for entering coding modules is the most complicated one in the NWB. The hard problem is how to provide a nice and easy-to-use interface for entering a tag set. Via the graphical user interface (GUI), the user must be able to describe:

1. the meta-data issues of the coding module;
2. the markup declaration, i.e. the tag set of the coding module.

Figure 7.1 shows how coding modules may be kept in folders, thus grouping together coding modules in the same area, such as gesture annotation. Of course, users may also choose not to keep coding modules together in folders like this. Users may create an empty coding module file anywhere or put an already defined one anywhere they like. However, when a coding module (whether empty or not) is selected for configuration, the interface will still be the same as the one shown in Figure 7.1.

Clicking on the Meta-data button enables the user to enter the coding module meta-data into the set of entries listed in Section 4. All entries are text entries and data is stored in tables in the same way as described for coded data above, i.e. with the possibility of linking. Figure 7.2 shows the meta-data interface for coding modules. Only the first five meta-data entries are shown. By default, the first eleven entries exemplified in Section 4.2 are listed under “type of information”. However, since there is no limitation on the number of rows which can be entered as meta-data in the NITE WorkBench database, users are free to include additional entries after the eleven recommended ones.

Clicking on the Modify button in Figure 7.1 will lead the user to the window shown in Figure 7.3. This is where the user enters a tag set.

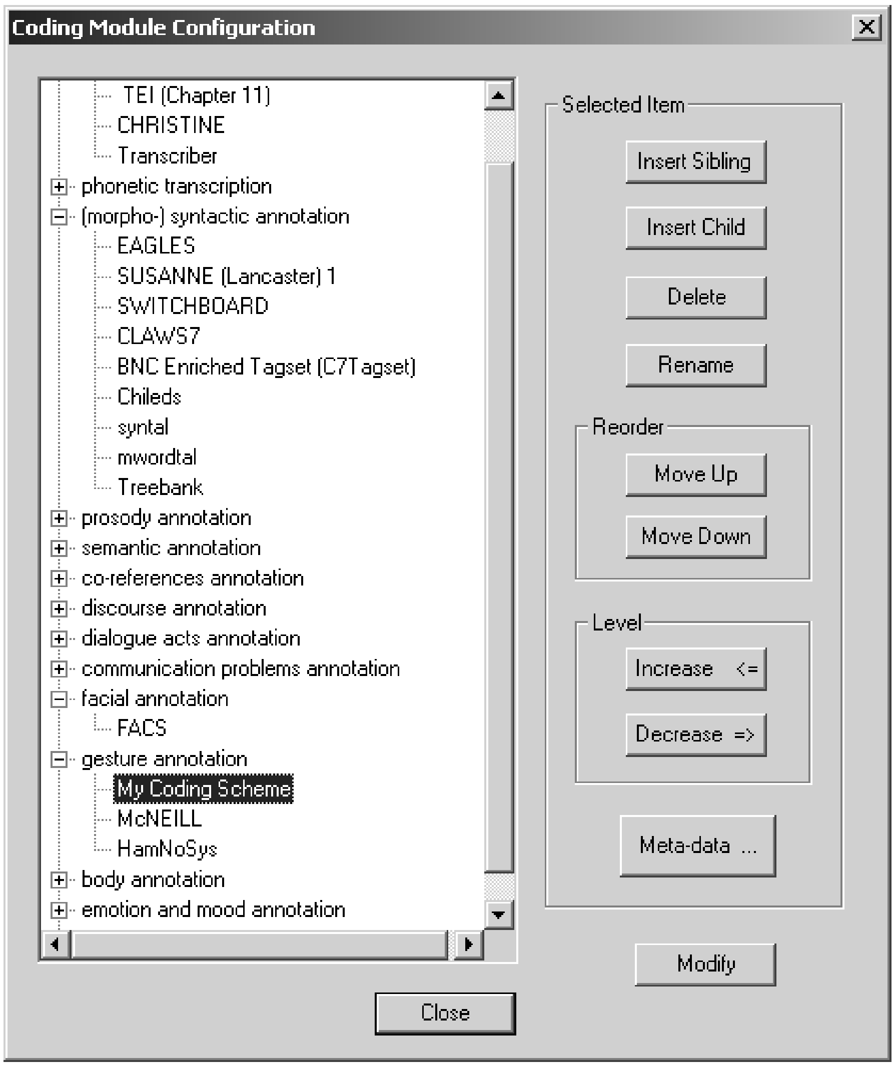
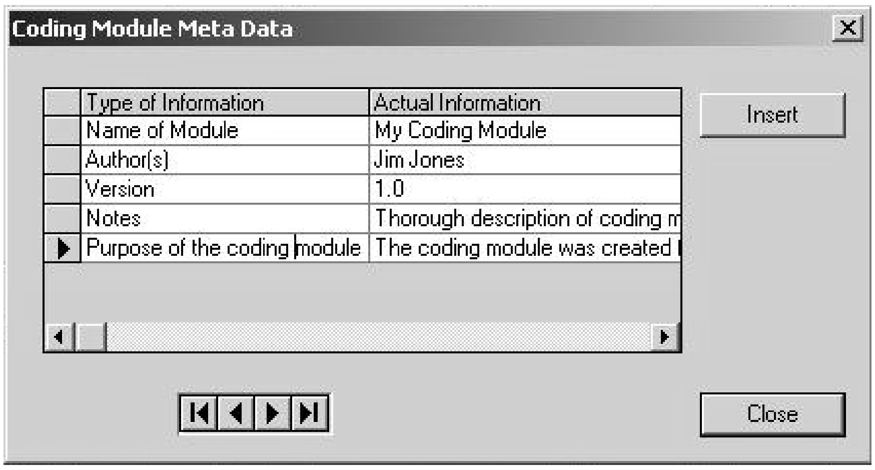


Figure 7.1. Coding modules are kept in folders, grouping together coding modules in the same area, e.g. gesture annotation. The buttons on the right allow the user to organise the files and folders shown in the window. The Meta-data button enables the user to add meta-data for a particular coding module. The Modify button allows the user to enter a tag set.



Type of Information	Actual Information
Name of Module	My Coding Module
Author(s)	Jim Jones
Version	1.0
Notes	Thorough description of coding m
Purpose of the coding module	The coding module was created

Figure 7.2. The meta-data dialogue box in the NITE WorkBench.

In order to specify a tag set for use in the NITE WorkBench, it is useful to first represent the coding scheme concepts hierarchically, e.g. as a set of tables or in a tree structure. An example of such a representation for prosody annotation – the ToBi coding scheme, [Klein et al. 1998] – is described in the following.

We distinguish between elements and attributes. Elements and attributes are also well-known concepts from TEI, SGML and XML. An element represents a certain type of phenomenon such as a particular phoneme, word, utterance, dialogue act, or communication problem we are interested in. Elements may have sub-elements, attributes, and relations to each other both within the current coding module and across coding modules. Attributes are assigned values during the coding, but for each attribute the type of its values must be defined in advance.

The phenomena of the ToBi coding scheme are organised in two levels in the representation below. At the first level we have three elements: prosodic boundaries, prosodic phenomena, and downstep. The element prosodic boundaries has five sub-elements each of which has an attribute used for visualisation and which has the value type shown in the table. The element prosodic phenomena has three sub-elements each of which again has sub-elements each with an attribute for visualisation and its value type. The element downstep simply has an attribute for visualisation. Users don't need to indicate an ID attribute since that is taken care of automatically by the system. Also, it is not necessary to indicate attributes for references to words or time points or similar. Referencing is handled via the user interface where the user, e.g., points out the word s/he wants to tag.

Element	**Sub-element**	**Attribute: value**
prosodic boundaries	clitic group boundary	tag: 0
	word boundary	tag: 1
	boundary with no tonal mark	tag: 2
	intermediate Phrase boundary	tag: 3
	intonative Phrase boundary	tag: 4

Table 1.1. Prosodic boundaries in ToBi.

Element	**Sub-elements**
prosodic phenomena	pitch accents
	boundary tones
	phrase accents

Table 1.2. Prosodic phenomena in ToBi.

Element	**Sub-elements**	**Attribute: value**
pitch accents	peak accent (high pitch accent)	tag: H*
	low accent (low pitch accent)	tag: L*
	scooped accent	tag: L*+H
	rising peak accent	tag: L+H*.
	downstepped accent	tag: H+!H*

Table 2.1. Pitch accents in ToBi.

Element	Sub-elements	Attribute: value
boundary tones	final low boundary tone	tag: L%
	final high boundary tone	tag: H%
	initial high boundary tone	tag: %H

Table 2.2. Boundary tones in ToBi.

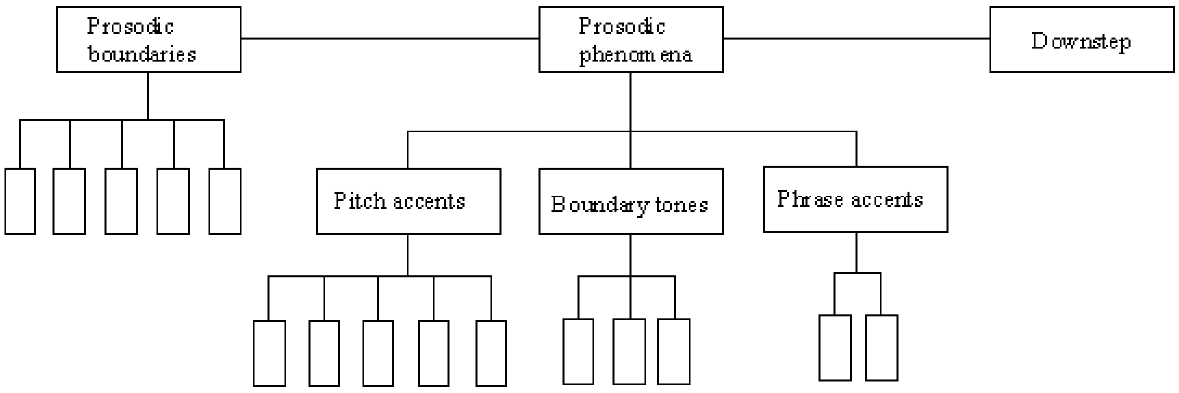
Element	Sub-elements	Attribute: value
phrase accents	low phase accent	tag: L-
	high phase accent	tag: H-

Table 2.3. Phrase accents in ToBi.

Element: downstep	Attribute: value
	tag: !

Table 1.3. Downstep in ToBi.

In summary, the above tables correspond to the tree representation below.



When the user has prepared a hierarchical description of the tag set, be it as a written document or otherwise, it becomes quite easy to enter the description into in the NITE WorkBench, cf. Figure 7.3. Attribute values are entered as shown in Figure 7.4.

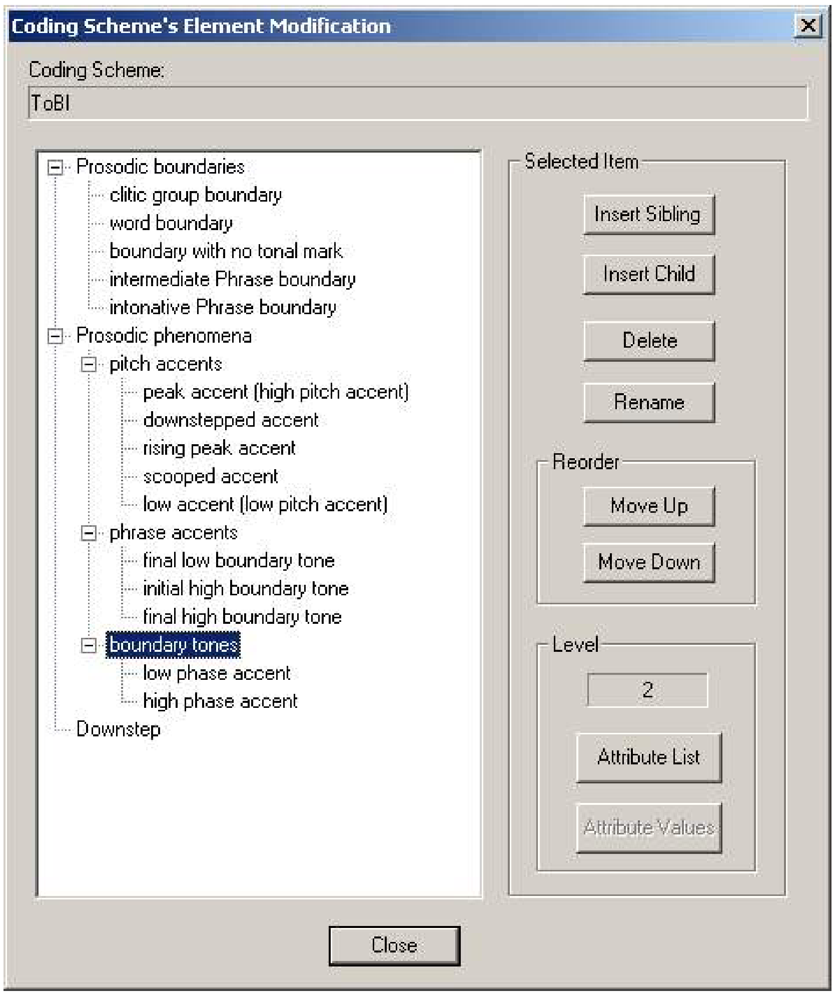


Figure 7.3. A tag set must be expressed in terms of elements and attributes.

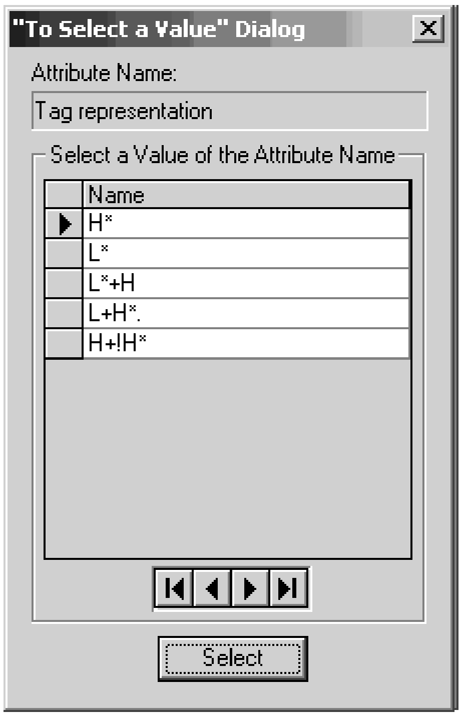

"To Select a Value" Dialog
Attribute Name:
Tag representation
Select a Value of the Attribute Name
Name
H*
L*
L*+H
L+H*.
H+!H*
Select

Figure 7.4. Entering attribute values into NWB.

7.3 Entering and storing meta-data

Meta-data for raw data and for coding files are entered in a window similar to the one shown for coding module meta-data in Figure 7.2. The difference is that the *type of information* column will show the NWB-recommended entries for raw data meta-data and coding file meta-data, respectively. As for coding module meta-data, the user will be able to add new meta-data entries.

7.4 Solving important issues via the database approach

As explained above, annotated data is stored in interrelated tables in the database. The structure of the database makes it straightforward to handle the issues mentioned in Section 6. It is not a problem to handle, e.g., overlaps and intersecting hierarchies, since there is nothing in the database structure which prohibits such representation. The example of intersecting hierarchies in Section 6 may be represented as illustrated in Figure 7.5. A tag always (indirectly) points to the start of a time point T_i. The tags need not physically be listed in any particular order relative to one another. They just have to point to the right words. By doing this they are of course indirectly ordered since the words are ordered in time.

Time point	Orthographic transcription	Annotation level X
1	John	tag A starts
2	likes	tag B starts
3	Mary	tag A ends
4		tag B ends

Figure 7.5. Intersecting hierarchies represented in a database.

This flexible database link structure also facilitates the representation of synchronous and asynchronous phenomena. It is not a problem that things are happening at the same time since there may be many links to the same cell.

Not only the codings but also the raw data will be linked to a time line. This enables time alignment of raw data and its different codings.

Cross-level and cross-modality coding involve references across levels of annotation (e.g. dialogue acts and miscommunication) or across modalities (e.g. speech and gesture). Again, the flexible database structure facilitates the linking. Any cell may in principle link to any other cell.

Long-range dependencies are represented via tags which belong together. Thus, e.g., if a long range-dependency involves two parts, these my be marked up with the tag A part 1 starts and ends and tag A part 2 starts and ends. Note that, in general, if the annotated sequence is a point rather than an interval, the starts and ends are not needed. In the case of the long-range dependencies we may then have tag A part 1 and tag A part 2.

8 The NITE XML Toolkit approach

We describe what metadata and coding information NXT stores, how it is stored, and how to work with it.

8.1 NXT storage for metadata and information about codings

In the NITE XML Toolkit, information for a corpus about its signals, codings, defined display and editing tools, and data views combining synchronized tools are stored in the corpus metadata file, along with a list of the observations in the corpus that details which codings have been deployed on each observation.

8.1.1 Raw data (unannotated signal)

A corpus is a set of observations all of which conform to the same basic format and have the same number of agents being observed, with the same basic roles. The following table shows how to fit some well-known corpus types into this categorization:

	AGENTS
Map task corpus:	giver,follower
Smartkom:	system,user
Wall Street Journal articles:	writer
Five person discussion:	1,2,3,4,5

Figure 8.1. Agent roles in different corpora.

For group discussion corpora of mixed size, the user must define agents for the maximum size and fail to use some of them for the observations with fewer people.

Each observation will have been recorded separately using some signal or set of signals. Signals can either be for a single agent (like a video trained exclusively on the route giver), or of the interaction as a whole (like an overhead video that captures the whole group, or at least part of it). All signals for the same observation are assumed to start at the same time. This can be achieved through pre-editing. Note that because there could be several video signals associated with the same corpus, any GVM (video overlay markup) needs to know which signal it applies to.

For all corpora, the metadata lists a set of observations, a set of agent roles, and a set of types of signals associated with each observation, identifying the types as being of an agent (in which case, there is one for each agent role per observation) or as being of an interaction (in which case, there is one for per observation).

For all corpora, we must specify what reserved attributes are used for ids, start and end times aligning tags to signal, and which agent a tag is associated with (if any). Ids must be unique in the XML file in which they appear, and to promote the possibility of different serializations, we strongly recommend that ids are unique in the entire corpus. Note that the types and legal appearance / non-appearance of all of these attributes is set. IDs are free strings that are unique in the corpus, not just the XML file, start and end times are numbers, and agent designators must be the name (role) of an agent. Time-aligned elements must have all three attributes; structural-layer elements must have ID and may have start and end times; featural-layer elements must have id but must not have start and end times. Layers are defined below.

In addition to this information, there is room for generic information about the corpus, like where to find documentation, and about individual observations - date of recording, condition the observation fits into within a planned design and so on. We show how this information can be included but don't make a serious attempt to catalog it since we don't expect to produce an interface that makes use of it.

8.2 Codes (tags, annotations)

Each observation may also have been coded, or annotated, with information that interprets the raw data.

Codes come in layers where all the codes share something about their meaning (for instance, a layer of things that make orthographic transcription, a layer of gestures). There are three types of layers:

1. Time-aligned layers are layers of codes that are directly aligned to signal by giving them start and end times (example: a layer of words, breaths, and silences that span an audio signal and make an orthographic transcription). Codes in a time-aligned layer can have attributes (example: part-of-speech for words, length for silence) and can contain text but not have children. One would ordinarily follow the TEI convention that any textual content represents speech content (words have text content, silences are empty) but this is up to the corpus designer.
2. Structural layers are layers of codes that have start and end times, but inherit them from codes on a time-aligned layer which are considered to be part of the structural code. (example: a layer of syntax where the codes are np, vp, and so on, and each code has as children words from a lower time-aligned layer).
3. Featural layers are layers of codes that are universal (that is, not timed) where the codes have named roles that point to other codes. Even if the code pointed to has timing information, it doesn't apply to the featural code.

Where the set of codes associated with a corpus fits into a set of structural layers which each point down from one to the next, possibly with one time-aligned layer at the bottom, then the data will fit into one XML file. This arrangement is called a *simple* corpus.

Otherwise we need a *standoff* corpus that splits the layers into *codings*, identifying them as codings for one agent (like gesture coding for a particular agent, in which case, there is one for each agent role per observation) or for the interaction (like codes for handshakes or for dialogue structure that gives the logical order of turns in the dialogue, in which case, there is one per observation). Each coding is a set of layers where any featural layer is on the top, any time-aligned layer is on the bottom, and there are structural layers in between where every structural code points only to codes in lower layers or in the top layer of another coding. For instance, in a standoff corpus there may be several structural layers that point to the same time-aligned layer. This is espcially true where corpora are coded for several different phenomema at once (e.g., gesture, dialogue structure, disfluency). Each coding gets serialized (stored) in one XML file.

Whether a corpus is simple or standoff is given in the type attribute of the corpus element in the metadata file.

For simple corpora, the grammar defining the set of layers and how they relate to each other is given as the content of the codings tag.

For standoff corpora, the metadata names the codings and gives a grammar for each one separately. Relationships among the files are allowed by the fact that a layer in one coding can point to a layer in a different coding.

For standoff corpora, there may additionally be a list of **ontologies** which are hierarchies, independent of an individual coding. These hierarchies may be pointed into by coding files, but the elements in the ontology files may not themselves point to anything outside the file. An example of an ontology would be a gesture type hierarchy which could be pointed into by a gesture coding file.

For standoff corpora, there may additionally be a list of **object sets** which are flat lists of elements independent of any individual coding. These may be pointed into by coding files, but the elements in the object set files may not themselves point to anything outside the file. Examples are lexicons that the words point to and lists of entities in the world, pointed to by referring expressions.

For standoff corpora, adhering to this structure is necessary because it is what the NOM uses. For more information about the structure, see the NOM API.

For simple corpora, violating this structure only means that data can't be validated against the layer model (but it could be validated against a DTD or schema) and one can't use OTABs to display it. Because the stylesheets for simple corpora use JDOM, a standard XML data model, they will still work for other single XML file structures.

8.2.1 Displays and editors

A display can not change the underlying data it represents; an editor can.

There are two applications within the NITE XML Toolkit that can produce displays and editors:

- OTAB - Observable Track Annotation Board
- NIE - NITE Interface Engine

OTAB produces displays based on the annotation board concept. NIE is a stylesheet based engine which is generally used to produce transcription views of the data. The output from both NIE and OTAB are referred to as styled displays or editors.

An OTAB is a time-aligned data editor based on the annotation board concept. OTABS can be used to edit the data by (1) changing the times of time-aligned tags [but not the times of tags that inherit their timing from other tags] and (2) adding new tags or deleting existing tags. For all corpora, the metadata lists a set of OTAB configuration files that can be applied to the codings to make an editor. (In the metadata files, this is called an annotation-spec; this name should probably be changed.) The OTAB configuration file specifies what the OTAB should look like when it is first displayed: the overall look-and-feel, which tiers to display and which to hide, and what colour schemes to use for the tiers. This is equivalent to the formatting part of an Anvil configuration file, but Anvil configuration files also have information about the codings that for NITE, are in the metadata file instead. No matter what configuration an OTAB has when it starts up, the user will be able to change it using menus provided on the OTAB.

An NIE (transcription-based) display or editor shows the structure of a set of layers in relation to the codes on the time-aligned layer at the base of the set of layers (conventionally some kind of transcription, but it could be a set of gestures, or so on.) For all corpora, the metadata lists a set of stylesheets that can be applied to the codings to make a transcription-based display or editor. For standoff corpora, each also specifies a list of codings it uses, and a list of codings it changes. This means we don't necessarily have to load and serialize everything, and is just an efficiency. The list is just a catalog that describes the stylesheet and tells where to find them (path and filename). Stylesheets for simple corpora use XPATH queries and JDOM manipulations to specify their editing behaviour. Stylesheets for stand-off corpora use our own query language and NOM manipulations.

In addition to these methods of constructing displays and editors, there are also the following types of displays and editors:

1. audio displays, which can be muted;
2. video displays, which can be muted;
3. GVM editors, which show a video, can be muted and which can, when used in conjunction with an OTAB, be used to display and edit graphical visual markup (GVM).

8.2.2 Data views

A data view consists of a set of displays and editors, all of which are registered with the data view's clock so that they are synchronized to show the same time.

For all corpora, the metadata file lists a set of data views, each of which in turn lists the displays and editors that they employ.

No matter what set of displays and editors a data view employs, the user can turn them off using the close button for the windows involved and turn new ones on using the menus.

8.2.3 Observations

Each corpus needs a list of the observations that together, form that corpus. For each observation, there is a list of the codings that have been deployed for that observation. Information about who performed a coding, when, and its status can also be stored here.

8.3 The NXT metadata format

This information is stored in the corpus metadata file, which is expressed in an XML format. Appendices A and B give examples for simple and stand-off corpora, respectively.

8.4 Working with this information

Part of the NITE XML Toolkit is an API through which files in the NXT Metadata Format can be loaded and the information extracted. This is the normal mechanism for working with the files. Because the API and our implementation are open source, and because the implementation is based on XML querying which is robust in the face of additions to the structure present in the file, if the current provision is insufficient for any particular project, extending it is an easy task. Similarly, the API and implementation can be extended to manipulate the metadata rather than just reading it. However, within the scope of this project we expect this metadata to suffice and for it to be written in a normal XML editor.

9 References

Bernsen, N.O., Cadée, N., Carletta, J., Dybkjær, L., Evert, S., Heid, U., Isard, A., Kolodnytsky, M., Lauer, C., Lezius, W., Noldus, L., Reithinger, N. and Vögele, A.: Lists of Requirements. Addendum to NITE Deliverable D1.1, 2002.

Broeder, D., Offenga, F., Willems, D. and Wittenburg, P.: EAGLES / ISLE Metadata Set for Multimedia/Multimodal Language Resources. ISLE Deliverable D10.2, 2002.

Carletta, J., Bernsen, N.O., Cadée, N., Dybkjær, L., Evert, S., Heid, U., Isard, A., Kolodnytsky, M., Lauer, C., Lezius, W., Noldus, L., Reithinger, N. and Vögele, A.: Software Specification and Workplan. NITE Deliverable D1.1, 2001.

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

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

Knudsen, M. W., Martin, J.-C., Dybkjær, L., Ayuso, M. J. M, N., Bernsen, N. O., Carletta, J., Kita, S., Heid, U., Llisterri, J., Pelachaud, C., Poggi, I., Reithinger, N., van ElsWijk, G. and Wittenburg, P.: Survey of Multimodal Annotation Schemes and Best Practice. ISLE Deliverable D9.1, 2002.

Serenari, M., Dybkjær, L., Heid, U., Kipp, M. and Reithinger, N.: Survey of Existing Gesture, Facial Expression, and Cross-Modality Coding Schemes. NITE Deliverable D2.1, 2002.

10 Appendix A: NXT Metadata for a Simple Corpus

```
<!-- 07 OCTOBER 2002
NXT METADATA EXAMPLE FOR SIMPLE CORPUS              JEAN CARLETTA
Metadata version 1.1; introduces the notion of views that combine
OTABS and stylesheets.

This is based on the Smartkom data circulated with Anvil.
-->

<corpus id="smartkom" description="Smartkom Corpus" type="simple">

    <!-- GENERIC CORPUS INFORMATION -->
    <!-- the external documents people can refer to -->

    <documents>
    </documents>

    <reserved-attributes>

        <identifier name="id"/>
      <starttime  name="start"/>
      <endtime    name="end"/>
      <agent      name="who"/>

    </reserved-attributes>

    <!-- LIST OF AGENTS -->

    <agents>
      <agent name="s" description="system"/>
      <agent name="u" description="follower"/>
    </agents>


    <!-- LIST OF SIGNALS -->

    <signals path="Data/signals/">

      <agent-signals>
             <!-- for this corpus, there aren't any -->
      </agent-signals>

      <interaction-signals>
          <signal name="interaction-video" type="video"
                format="AVI" extension="avi"/>
      </interaction-signals>

    </signals>


    <!-- CODINGS/{agent|interaction}-coding/coding-file for standoff;
         just go straight to the content model for a simple corpus -->

    <codings path="Data/xml/">
                <time-aligned-layer name="timed-unit">
                   <code name="utterance"/>
            </time-aligned-layer>
    </codings>
```

```
     <!-- Styles define ways of constructing windows that hold
          transcription or annotation based displays and coding
          interfaces (editors).  It's important whether they allow the
          data they show to be edited or not, because that affects
          whether or not the user can save.  -->

     <styles path="Data/xsl/">
         <style name="smartkom-trans" description="basic transcription
display" type="display" application="nie" extension=".xsl"/>
         <style name="smartkom-pos" description="part-of-speech display"
type="display" application="nie" extension=".xsl"/>
         <style name="smartkom-game" description="dialogue structure
editor" type="editor" application="nie" extension=".xsl"/>

         <style name="gesture-only" description="gesture OTAB"
type="editor" application="otab" extension=".oc"/>
         <style name="everything" description="full OTAB" type="editor"
application="otab" extension=".oc"/>
     </styles>


     <!-- This defines the preconfigured views of the data.
       -->
     <views>

         <!-- This view is a display that only shows a video and a
transcription. -->

         <view description="transcription" type="display">
             <!-- Give a video window with the interaction-video for the
observation in it.
                  It shouldn't have the gvm loaded for it because we aren't
going to use it.
               -->
             <!-- If we had a separate, better audio recording for the same
observation, then
                  we might want to put up an audio window for that, and
turn the sound in the
                  video off.  This happens in some corpora.
               -->
             <video-window nameref="interaction-video" sound="yes"/>
             <!-- Give a simple transcription display as produced by the
stylesheet smartkom-trans.-->
             <styled-window nameref="smartkom-trans"/>
         </view>

         <!-- This view is a display with a gesture OTAB and a video.  -->
         <view description="gesture">
           <video-window nameref="interaction-video" sound="yes"/>
             <styled-window nameref="gesture-only"/>
         </view>

         <!-- This view is a display that shows a full OTAB and a styled
view of the dialogue structure.
           -->
         <view description="gesture">
           <video-window nameref="interaction-video" sound="yes"/>
             <styled-window nameref="everything"/>
             <styled-window nameref="smartkom-games"/>
         </view>
     </views>



     <!-- a list of the observations in a corpus. -->

     <observations>

          <observation name="smartkom1">
                <!-- this contains a list of codings present for standoff
corpora, and nothing
                      for simple corpora
                 -->

          </observation>
          <observation name="smartkom2">

          </observation>
          <observation name="smartkom3">

          </observation>
          <observation name="smartkom4">

          </observation>
          <observation name="smartkom5">

          </observation>

   </observations>

</corpus>
```

11 Appendix B: NXT Metadata for a Stand-off Corpus

```
<!-- NXT METADATA EXAMPLE FOR A STANDOFF CORPUS
     JEAN CARLETTA AND JONATHAN KILGOUR
     7th October 2002
-->

<corpus id="maptask" description="Map Task Corpus">

    <!-- GENERIC CORPUS INFORMATION -->

    <reserved-attributes>
        <identifier name="nite:id"/>
      <starttime  name="nite:start"/>
      <endtime    name="nite:end"/>
        <agent      name="who"/>
    </reserved-attributes>

    <reserved-elements>
        <pointer name="nite:pointer"/>
      <child  name="nite:child"/>
      <text  name="nite:text"/>
    </reserved-elements>


    <!-- the external documents people can refer to -->
    <documents>
      <document description="Lang and Speech paper describing corpus"
                        URL="http://www.cogsci.ed.ac.uk/"/>
    </documents>

    <observation-variables>
         <variable name="eye-contact" type="enumerated">
             <value>no eye</value>
             <value>eye</value>
         </variable>
         <variable name="familiarity" type="enumerated">
             <value>familiar</value>
             <value>non-familiar</value>
         </variable>
    </observation-variables>

    <!-- AGENTS -->
    <agents>
      <agent name="g" description="giver"/>
      <agent name="f" description="follower"/>
    </agents>


    <!-- SIGNALS -->
    <signals path="/home/corpora/MapTask/CDs">
      <agent-signals>
          <signal name="audio" type="audio" format="mono au"
                  extension="au"/>
      </agent-signals>

      <interaction-signals>
          <signal name="interaction-audio" type="audio"
                format="stereo au" extension="au"/>
          <signal name="interaction-video" type="video"
```

```
              format="avi" extension="avi"/>
    </interaction-signals>
  </signals>


  <!-- ONTOLOGIES -->
  <!-- ontologies are static hierarchies e.g. a gesture ontology  -->
  <ontologies path="Data/xml">
      <ontology-file name="gtypes" description="gesture ontology">
        <!-- this is recursive, but doesn't ground-out on anything -->
        <featural-layer name="gesture-type-layer" recursive="true" >
          <code name="gtype">
              <attribute name="type" value-type="string"/>
          </code>
        </featural-layer>
    </ontology-file>
  </ontologies>

  <!-- OBJECT SETS -->
  <!-- Corpora may also be associated with object sets like a list of
     referents or similar. The syntax will be:  -->
  <object-sets path="Data/xml">
        <object-set-file name="xyz" description="abc">
        <featural-layer name="object-layer">
          <code name="obj">
              <attribute name="type" value-type="string"/>
          </code>
        </featural-layer>
      </object-set-file>
  </object-sets>


  <!-- CODINGS -->
  <codings  path="/home/corpora/MapTask/codings">
    <agent-codings>
        <!-- most basic coding for this corpus per agent is
          transcription -->
        <coding-file name="transcription">
            <time-aligned-layer name="timed-unit">
              <code name="tu">
                  <attribute name="utt" value-type="number"/>
              </code>
              <code name="noi">
                  <attribute name="type" value-type="enumerated">
                  <value>breath</value>
                  <value>inbreath</value>
                  <value>outbreath</value>
                  <value>lowamp</value>
                  <value>nonvocal</value>
                  <value>cough</value>
                  <value>lipsmack</value>
                </attribute>
              </code>
              <code name="sil"/>
          </time-aligned-layer>

          <structural-layer name="word" points-to="timed-unit">
              <code name="word"/>
          </structural-layer>
        </coding-file>
```

```
          <coding-file name="syntax">
            <structural-layer name="part-of-speech" points-to="word">
                <code name="n"/>
                <code name="at"/>
            </structural-layer>

            <structural-layer name="constituents"
                              recursive-points-to="part-of-speech">
                <code name="np"/>
                <code name="vp"/>
            </structural-layer>
          </coding-file>

          <coding-file name="move">
            <structural-layer name="move" points-to="word">
                <code name="instruct"/>
                <code name="explain"/>
            </structural-layer>
          </coding-file>

          <coding-file name="reference">
            <structural-layer name="reference" points-to="constituents">
            <!-- should we be able to specify that references must
                  point to words that are NPs? -->
                <code name="deixis"/>
                <code name="pronoun"/>
            </structural-layer>
          </coding-file>

          <coding-file name="gesture">
            <time-aligned-layer name="gesture-phase">
                  <code name="prep"/>
                  <code name="resolution"/>
            </time-aligned-layer>
            <structural-layer name="gesture-phrase" points-to="gesture-
phase">
                  <code name="baton"/>
            </structural-layer>
          </coding-file>

          <coding-file name="deixis">
            <featural-layer name="pointings">
               <code name="pointing-event">
                  <pointer role="gesture-trace" target="gesture-phrase"
number="1"/>
                  <pointer role="speech-trace" target="reference"
number="1"/>
               </code>
            </featural-layer>
          </coding-file>
      </agent-codings>

      <interaction-codings>
          <coding-file name="game">
              <featural-layer name="game-layer">
                <code name="game">
                    <role type="move-in-game" points-to="move" number="+"/>
                </code>
            </featural-layer>
          </coding-file>
      </interaction-codings>
```

```
     </codings>

     <styles path="/home/corpora/MapTask/styles">
         <style name="transcription" extension=".xsl" application="nie"
description="basic syntax display" type="display">
          <coding-ref name="transcription"/>
          <coding-ref name="syntax"/>
          <ontology-ref name="gtypes"/>
       </style>

       <style name="move-coder" extension=".xsl" application="nie"
description="move coder" type="editor">
          <coding-ref name="transcription"/>
          <coding-ref name="dialogue-structure"/>
       </style>

         <!-- an OTAB that shows words and moves -->
         <style name="maptask-dialstruct" extension=".oc"
application="otab" description="dialogue structure" type="display">
          <coding-ref name="word"/>
          <coding-ref name="move"/>
       </style>

         <!-- an OTAB that shows everything -->
         <style name="maptask-all" extension=".oc" application="otab"
description="everything"  type="editor" >
          <coding-ref name="word"/>
          <coding-ref name="move"/>
          <coding-ref name="transcription"/>
          <coding-ref name="syntax"/>
          <coding-ref name="dialogue-structure"/>
          <ontology-ref name="gtypes"/>
       </style>
     </styles>

     <views>
         <view description="game coder" type="editor">
             <!-- Give a video window with the interaction-video for the
observation in it.
                  Turn off the sound because we have a better audio signal.
               -->
             <video-window nameref="interaction-video" sound="no"/>
             <audio-window nameref="interaction-audio" sound="yes"/>
             <styled-window nameref="move-coder"/>
         </view>
         <view description="basic transcription" type="display">
             <audio-window nameref="interaction-audio" sound="yes"/>
             <styled-window nameref="transcription"/>
         </view>
         <view description="time aligned moves" type="display">
<!--              <gvm-window nameref="interaction-video" sound="yes"/> -->
             <styled-window nameref="maptask-dialstruct"/>
         </view>
     </views>

     <observations>
          <observation name="q1ec1" eye-contact="eye"
familiarity="familiar">
            <codings>
                <coding name="transcription" status="final"
                              coder="cathy" date="23sep98"/>
                <coding name="move" status="final"
                              coder="gwyneth" date="12oct01"/>
                <coding name="syntax" status="draft"
                              coder="david" date="15oct01"/>
            </codings>
          </observation>
          <observation name="q1ec2" eye-contact="eye" familiarity="non-
familiar">
            <codings>
                <coding name="transcription" status="final"
                              coder="cathy" date="24sep98"/>
                <coding name="move" status="final"
                              coder="gwyneth" date="12oct01"/>
            </codings>
          </observation>
          <observation name="q1ec3" eye-contact="eye" familiarity="non-
familiar">
            <codings>
                <coding name="transcription" status="final"
                              coder="cathy" date="25sep98"/>
                <coding name="move" status="final"
                              coder="gwyneth" date="12oct01"/>
            </codings>
          </observation>
          <observation name="q1ec4" eye-contact="eye" familiarity="non-
familiar">
            <codings>
                <coding name="transcription" status="final"
                              coder="cathy" date="25sep98"/>
                <coding name="move" status="final"
                              coder="gwyneth" date="13oct01"/>
            </codings>
          </observation>
          <observation name="q1ec5" eye-contact="eye" familiarity="non-
familiar">
            <codings>
                <coding name="transcription" status="final"
                              coder="cathy" date="26sep98"/>
                <coding name="move" status="final"
                              coder="gwyneth" date="13oct01"/>
            </codings>
          </observation>
   </observations>
</corpus>
```