A Visual Interface for a Multimodal Interactivity Annotation Tool: Design Issues and Implementation Solutions

Mykola Kolodnytsky, Niels Ole Bernsen, Laila Dybkjær
NISLab, University of Southern Denmark, Campusvej 55, 5230 Odense M, +45 65 50 35 51
{mykola, nob, laila}@nis.sdu.dk

ABSTRACT

This paper discusses the user interface design for the NITE WorkBench for Windows (NWB) which enables annotation and analysis of full natural interactive communicative behaviour between humans and between humans and systems. The system enables users to perceive voice and video data and control its presentation when performing multi-level, cross-level and cross-modality annotation, information visualisation for data coding and analysis, information retrieval, and data exploitation.

Categories and Subject Descriptors

H5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems – *audio input/output, video*. H5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical User Interfaces (GUI), interaction styles*.

General Terms

Performance, Design, Human Factors.

Keywords

Data annotation tools, data visualisation, interface design.

1. INTRODUCTION

A basic precondition for building natural interactive systems is consolidated theoretical knowledge about how humans communicate with one another or with a machine through modalities such as speech, gesture, gaze, facial expression, object manipulation, etc. To gain this knowledge, researchers first collect high-quality audio and video *corpora*. They then annotate the corpus data based on *coding schemes* (CS) which often represent consistent and complete classifications of different classes of phenomena found in human communication. Data annotators need coding tools which make it as easy as possible to code new corpora and do a number of related activities. This paper describes a coding tool developed for these purposes and called the NITE Workbench for Windows, or NWB. The tool has been developed in the European project Natural Interactivity Tools Engineering (nite.nis.sdu.dk) and aims to eventually support full general-purpose coding of natural interactivity corpora.

2. 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 [1,2].

Cross-level coding. Annotation of a particular modality, such as speech, may be done at different levels, such as orthographic transcription, prosody, or morpho-syntax. A *level of annotation* is a 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 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 in order to capture coordinated human communication behaviours. 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 in time while group interactions involve at least two parties performing a shared action, such as 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.

Long-range dependencies. A phenomenon may involve long-range dependencies 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, i.e. we might want to display long-range dependencies through symbolic tags or by using colour coding.

Synchronous and asynchronous phenomena. Phenomena to be tagged 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. The annotation tool must enable alignment of raw data and different annotations of the raw data to the same timeline. The timeline provides a common anchor point in time for transcriptions and other codings, and helps sort out the sequentialisation of the markup. The timeline may be shown as a line (analogue view) or as a sequence of symbolic time tags.

3. USER INTERFACE FUNCTIONAL REQUIREMENTS SPECIFICATION

Based on the full NWB requirements specification [2,3], the core user interface (UI) functional requirements can be combined into the following six groups:

1. To work with an annotation project file, i.e. create, open, save, print or import into XML file components of a project. The project file consists of a reference to a raw data file, references to coding files containing annotations (including transcriptions) of the raw data, and a reference to meta-data for each raw data file and each coding file.

2. To enable users to specify coding schemes. To specify a coding scheme means to create a new UI component for an annotation scheme, or to modify structure and/or data contents of an annotation scheme already entered into the NWB.

3. To control raw data audio/video files, i.e. to: visualise and play video files created in current standard formats, such as *.mpg, or *.avi; listen to the audio track created in current standard formats, such as *.mp3 or *.wav; manage the raw data windows (show, hide, move, resize); graphically represent the acoustic information (wave-form, spectrum, etc.); have a visible timeline, zoom in/out on the timeline, navigate back and forth in the raw data based on the timeline, i.e. scroll, go step-by-step along the timeline, jump to the beginning or end of the timeline; display the value of timeline units, such as seconds, frame numbers for video data, or milliseconds for video and audio data; synchronize the displaying of data in different windows on the basis of the common timeline; etc.

4. To support annotation using coding schemes. This means to use a special-purpose (i.e. special format) file created as part of a project and containing the coding information. We call this file a coding file (or markup file). The coding file has to reflect the timeline of events in the raw data. We call it "the common timeline" or merely "the timeline". This also means to: select (i.e. indicate) the time point or time interval onto the timeline; select an appropriate coding scheme; select a tag from a list of tags for the coding scheme presented in a palette; and to edit the coding file, i.e. insert or delete tags from the coding scheme.

5. To visualise information in a customised way. This means to provide a window or windows (or its part like a split pane) for displaying the contents of a coding file. We shall call such a window an "annotation panel". It also means to display several annotation panels at the same time and to control the appearance of a panel on the screen, i.e. to show, hide, move, resize it, and change some attributes of it like the title, background colour, etc. Each annotation panel should correspond to a certain class of phenomena to be coded. Since a class of phenomena could be a hierarchy of categories and values, it should be possible to reflect this structure using a set of "tiers" ("bands", "layers"), each of which aims to display a particular subset of the categories. It should also be possible to: show the location (a "cursor") of the current position on the timeline; synchronise the cursors in different windows: as soon as the cursor moves in one window, it moves in the other ones and appears within the same timeline segment; view annotations and transcriptions at different levels of resolution in each annotation panel (zoom in/out along timeline, scroll along timeline); and visualise the result of the coding, i.e. provide various way of displaying the tags inserted onto the timeline within the annotation panel. The visualisation could be done using either a special-purpose window (called an annotation panel here), and/or directly on the window displaying the raw data (e.g. graphical video markup, GVM) or on the audio data wave-form, or the visualised and customised tags could be inserted onto the timeline using the annotation panel.

6. To enable information extraction and analysis of annotated data. This means to extract arbitrary parts of data by using, e.g., SQL queries, and to export, e.g., XML files to be used in external software for statistical descriptions and analysis of data.

4. MAPPING THE REQUIREMENTS INTO GUI PRIMITIVES

Given the core user interface functional requirements described above, the elaboration of GUI modules has been performed by answering questions like the following: "What does a user *want* to have in the NWB GUI to be presented?", "What kind of graphical primitives *can be used* to implement those requirements?", or "What type of GUI architecture and underlying framework should be chosen, if any?"

The most common GUI architectures suggested by "The Windows 95 Interface Guidelines" are the Single Document Interface (SDI) model and the Multiple Document Interface (MDI) [4]. Although MDI provides useful conventions for managing a set of related windows, it is not the only means of supporting task management. Some of its window management techniques can be applied in alternative designs. The *workspaces*, *workbooks*, and *projects* models are examples of possible design alternatives. They present a single window design model but in a way that preserves some of the window and task management benefits found in MDI [4].

In addition to the models mentioned above, The Rogue Wave Stingray Studio 2002 Objective Toolkit offers yet other MDI alternatives and enhancements, i.e. Multiple Top-level Interface (MTI), Floating Document Interface (FDI), and Workbook Document Interface (WDI) [5].

None of the existing GUI standard architectures entirely meets the requirements specified above. We have therefore designed the NWB GUI as a mixture of standard interface components as shown in the architecture in Figure 1. The GUI interface constructs described in Section 5 are illustrated with screenshots.

The model of the NWB GUI can be classified as a single document interface with multiple windows views (SDI MV). The document is based on a database with some additional data serialised into a file. The views are:

– a (new) child window which is like an MDI-child window but corresponding to the same document;

– a (new) tabbed view in the same child window.

Each tabbed view represents information either in tabular (symbolic) or in analogue (graphical) view. In addition, graphical views can be presented using either the horizontal or the vertical timeline.

Usually, each view can have a split window, i.e. each can be split into two panes with appropriate scrolling within it. The pane, i.e. each part of the split view, represents the same type of timeline as the view has. The only benefit of a split window, then, is to have an additional scrolled view that presents another part of the same document. Since, from an implementation perspective, the split window is a dynamical one, each view's pane is of the same C++ view class. Here we have a development tool constraint: the MS MFC library does not provide, directly, any other way of dynamical split window creation. Another constraint is that it is only possible to deal with two panes of the same type once, for instance, either two horizontal or two vertical ones, but it is not possible to have a horizontal and a vertical timeline view in different panes of the same view.

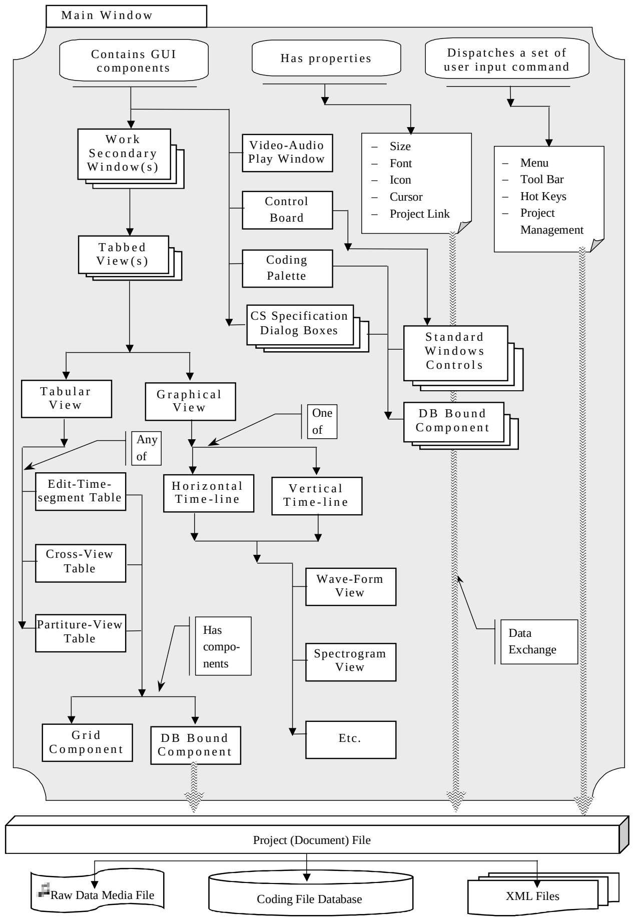

Figure 1. The NWB GUI architecture and ontology.

Bearing such problems in mind, we decided to embed a grid into the tabular view. The grid simulates the dynamical split window, the grid view can be customized, and the grid contents can be made database-aware. In addition to the child windows with views as described, there are a number of window components of another functional type: a video data window, a control board, a coding scheme palette. There is also a set of dialog-windows where the user can specify coding scheme structure and contents in order to enter a new coding scheme. The dialogues and palettes have a rich collection of different controls in them, such as static text fields and group boxes, buttons, tree view controls, combo boxes, spin boxes, slider, tab control, image, and database bound grid controls.

5. NWB GUI WALKTHROUGH

Below, we briefly describe the workflow in the NWB user interface and illustrate the solutions chosen for GUI component implementation.

When a project file or raw data file (video or sound) is opened, the NWB displays the Video Data window, the initially selected default view window (tabular or wave-form) and the Control Board window. The Video Data window and the Control Board can be hidden or shown again from the menu at any time. The user can also create several different child windows with different tabbed views in them. A sample screen shot of windows/views management is shown in Figure 2.

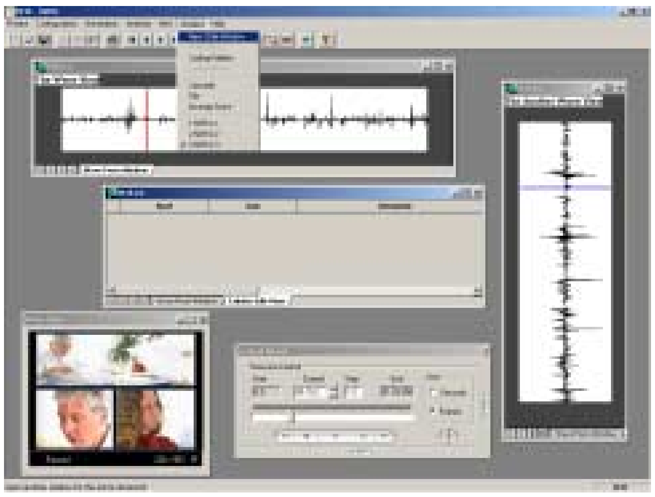

Figure 2. NWB windows/views management.

Since the NWB is a multi-coding scheme tool, it provides a way to define and organise, first of all, the coding schemes a user would like to include in the tool. Each coding scheme can be presented as a hierarchical set of elements some or all of which may have a list of attributes. To facilitate coding scheme entry, there is a kind of visual editor in the NWB. A workflow example of using the visual editor is shown in Figure 3.

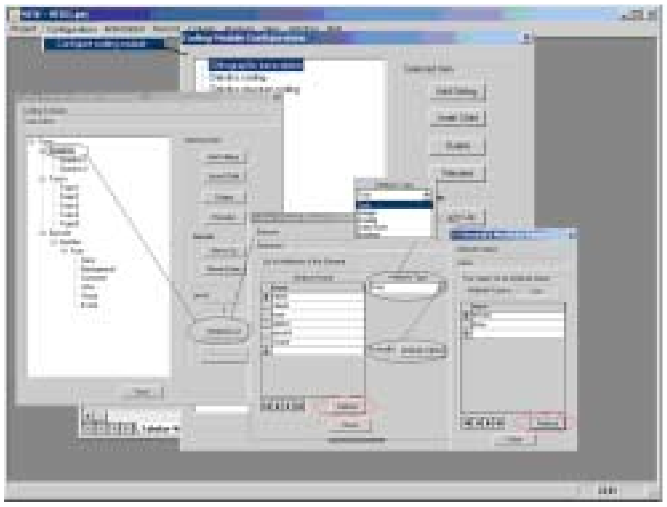

Figure 3. A coding scheme and its elements/attributes specification.

If the user wants to define attributes for an element in the coding scheme elements hierarchy, the user first selects the element in the tree-view control and then clicks on the Attribute List button as shown in Figure 3. This launches another dialogue window in which one can enter the attribute name and select an attribute type from the type list: text, integer, double, etc. Once an element attribute is validated using the refresh button, one can assign possible values for the attributes using the Attribute Values button which opens a dialogue window into which the values can be entered. The entered data will be stored in the project database.

The coding scheme illustrated in Figure 3 can be used for orthographic transcription of the spoken dialogue contributions of each speaker shown in the video window in Figure 4. The annotation procedure includes the following steps, cf. the numbered labels in Figure 4:

– to annotate a segment, position the time control at the beginning of the segment and select the element you want to annotate, cf. labels (1) and (2);

– then click on the insert coding segment button (3). The start and end time marks will appear automatically in the transcription tab;

– position the time control at the end of the segment, select the record, and click on the insert time-segment end button (4) (5).

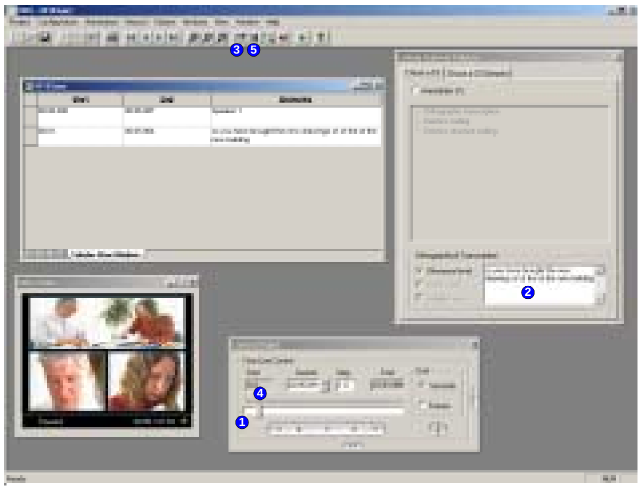

Figure 4. Utterance level transcription procedure.

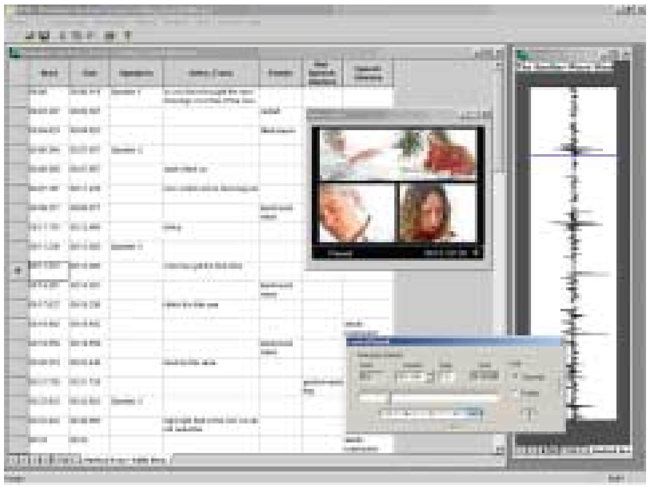

Figure 5. Cross-table view as a result of an SQL-query.

Figure 5 shows how annotated data can be visualised after one has annotated and/or transcribed it. The first two grid columns correspond to the annotated time segments. The following columns correspond to elements of two different coding schemes. Thus, Columns 3-4 include speaker references (Column 3) and orthographic transcription per speaker (Column 4). The following three columns 5-7 include events like background noise or pause, non-speech deictic events, such as pointing gestures, and speech deictic tags, such as for the deictic expression “this”, respectively. The number of grid columns is variable; it changes dynamically according to which number of coding scheme elements have actually been used in the annotation.

It is also worth noting here that any data visualisation and customisation is performed by underlying software components via the use of SQL queries. This provides reliability, good performance and flexibility of data mining and retrieval from annotated corpora.

6. CONCLUSION AND FUTURE WORK

The system description presented in this paper demonstrates that building an easy-to-use, general-purpose tool for the annotation and analysis of natural interactivity data is a complex task which needs to be staged in a number of development steps, each of which involves extensive studies and interface structure design.

Having performed the development steps, ongoing work addresses user testing, evaluation, and analysis of user feedback. Substantial user evaluation of the NWB forms the basis for extending the tool functionality.

Major planned functionality extensions include: the addition of an analogue, non-symbolic/tabular coding view in which segmented and tagged phenomena are viewed along with the timeline, enabling direct perception of temporal relationships; facilities for complex structure coding, in which phenomena tagged at different levels but representing coordinated human communication behaviour, can be linked, visualised and tagged as behavioural clusters; and plug-ins of specialist modules, such as new speech signal graphical representations like pitch contour view and spectrogram.

7. ACKNOWLEDGMENTS

We gratefully acknowledge the support of NITE by the European Commission’s Human Language Technologies Programme.

8. REFERENCES

[1] Bernsen, N. O., Dybkjær, L., and Kolodnytsky, M.: The NITE Workbench for annotating natural interactive behaviour. In: P. Paggio, K. Jokinen, and A. Jönsson (Eds.): Proceedings of the 1st Nordic Symposium on Multimodal Communication. CST, Copenhagen, Denmark, 2003, 155–168.

[2] Bernsen, N. O., Dybkjær, L., and Kolodnytsky, M.: An interface for annotating natural interactivity. In J. v. Kuppevelt and R. W. Smith (Eds.): Current and New Directions in Discourse and Dialogue, Dordrecht: Kluwer 2003, 35–62.

[3] Dybkjær, L., Bernsen, N.O., Carletta, J., Evert, S., Kolodnytsky, M. and O’Donnell, T.: The NITE Markup Framework. NITE Report D2.2. NISlab, Odense, Denmark 2002.

[4] The Windows 95 Interface Guidelines for Software Design. Online MSDN Library.

[5] Objective Toolkit User's Guide. Version 8.0. Rogue Wave Stingray Studio 2002 Version 2 Online Documentation.