ANVIL
A Generic Annotation Tool for Multimodal Dialogue

Michael Kipp

Graduate College for Cognitive Sciences
University of the Saarland, Germany
kipp@dfki.de

Abstract

Anvil is a tool for the annotation of audiovisual material containing multimodal dialogue. Annotation takes place on freely definable, multiple layers (tracks) by inserting time-anchored elements that hold a number of typed attribute-value pairs. Higher-level elements (suprasegmental) consist of a sequence of elements. Attributes contain symbols or cross-level links to arbitrary other elements. Anvil is highly generic (usable with different annotation schemes), platform-independent, XML-based and fitted with an intuitive graphical user interface. For project integration, Anvil offers the import of speech transcription and export of text and table data for further statistical processing.

1. Introduction

Recently, *multimodality* has become a crossroads where many disciplines meet. Multimodality refers to the merge of various sources of information, of multiple channels of communication, auditory (words, prosody, dialogue acts, rhetorical structure) and visual (gesture, posture, graphics).

In human-computer interface design, a multimodal interface is the natural extension of spoken dialogue systems where the user can communicate with speech and gesture and, in return, gets multimodal output. Annotation of multimodal dialogue on audiovisual media (video) is crucial for case-studies, training (machine learning), testing and evaluation.

In quite a different area, we find similar needs. Psychologists, ethnologists and anthropologists have long been concerned with the systematic exploration of human behavior, specifically the relation between nonverbal behavior and speech [1]. After early work with VCRs and special hardware, research is moving toward computer-supported investigation.

A third group is related to both of the mentioned fields. Researchers of *embodied synthetic characters* aim at interfacing with the user in a more efficient, natural and entertaining fashion [3]. Agent interfaces are promising for application in education, sales and presentation [2] but need more *empirical* investigation of human gesture and speech to make the agents' actions more "life-like" and believable.

Anvil[1] (Annotation of Video and Language) was developed with these needs in mind. In particular, it was designed to accommodate a large number of different schemes that deal with, e.g., prosody, syntax, dialogue acts, utterance/turn segmentation and rhetorical segments, and also gesture, posture and other behavioral units. Speech transcription is *not* supported and must be imported (Sec. 5).

[1] Anvil is free for research purposes. For download information visit http://www.dfki.de/~kipp/anvil

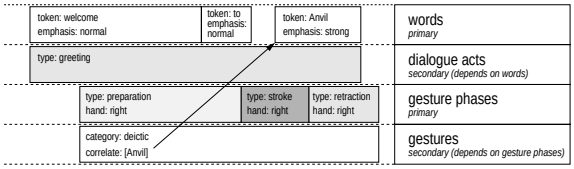

Figure 1: *Four sample tracks with elements.*

Anvil version 2.5 runs on any platform that accommodates Java2 (e.g. Sun's JDK 1.3) and Java Media Framework 2.0 (or higher). It was developed on a PentiumIII 500MHz under Windows98/2000/NT. Anvil requires the Java-based XML package Xerces-J[2] to be installed.

This paper first presents the basic generic framework of Anvil without referring to any specific annotation scheme. It will then proceed by explaining how to work with the tool, i.e. how the user interface is structured and how a specific annotation scheme can be implemented. Essential import/export functionality plus the concept of a project manager is introduced thereafter, and finally, the paper closes with a review of related work, a summary and plans for the future.

2. Basic Framework

This section explains Anvil's skeletal structure whereon a specific scheme must be fitted. How to customize Anvil with your specific scheme will be then described in Section 4.

Anvil, like many other tools, offers annotation on multiple layers (see Fig. 1), called *tracks*, say $T_1, \ldots, T_n$. Examples of tracks are: words[3], dialogue acts, gestures, postures shifts. During annotation the coder adds *elements* to a track. If these elements are anchored in time points for beginning and end the hosting track is called *primary*. This is the case for words. If elements of track T_i are defined by a sequence of elements in track T_j, e.g. a dialogue act as a sequence of words, track T_i is called *secondary*. Elements of T_i are said to *depend* on elements of T_j which is important when removing elements of T_j. Also, note that start/end time of *secondary* elements is defined as the start time of the first element and end time of the last element respectively. This definition is recursive as a secondary tracks can depend on yet another secondary track[4].

Each element holds a number of attribute-value pairs. Each attribute has a type that defines the range of possible values.

[2] visit xml.apache.org

[3] i.e. orthographic transcription

[4] Note that, to avoid cycles, we place the restriction that if T_i depends on T_j then $j < i$.

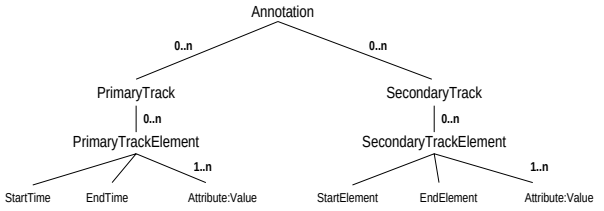

Figure 2: *Anvil's annotation object model.*

Anvil's attribute types are: String, Boolean, ValueSet and MultiLink. The latter two need explanation. A ValueSet is a user-defined list of strings. A MultiLink is a sequence of *cross-level pointers* to elements (excluding the currently edited element). This is the most powerful kind of annotation and, to my knowledge, not realized in any other tool. Formally, for an element e of track T_i, if e has an attribute a_k of type MultiLink then a_k can contain the sequence $(b_1, \ldots, b_p)$ of pointers where b_j denotes a pointer to an arbitrary element of an arbitrary track, excluding e itself (the arrow in Fig. 1 indicates such a pointer for the attribute "correlate" in track "gesture phrases"). Such pointers can be used (as in the example) to link up gestures with speech correlates (cross-level) or, e.g., to link up initiative-response pairs on the dialogue/speech act layer (within-level), exploiting the fact that elements need *not* be adjacent. *Co-reference* markup is yet another application.

Anvil's software design is object-oriented, understanding the annotation of a video as an annotation *object* containing itself track objects and so on. Fig. 2 shows the object model of Anvil, called the *logical layer* in [6]. At this level it is thinkable to offer an interface API (Java) where foreign programs can load, store and access Anvil annotations by direct class access.

Annotated data is stored into a *single* XML file. As opposed to other projects [9][4] we argue for a single file because it makes data more compact (no reference to file paths) and less susceptible to inconsistency caused by deleting/renaming files. Since nowadays it is easy to write XML file processing routines, multiple files (e.g. one file per track) do not facilitate processing any more. References to track elements can be uniquely specified within one file by track name plus index number.

3. User Interface

Anvil comes with a graphical front-end as shown in Figure 3. In the upper center you see the video window with the familiar VCR control panel (including frame-wise stepping and slow-motion playback). Below is the *annotation board* where all the coding takes place. The different track/group[5] names are displayed on the left side, the actual tracks take up the rest of the window, elements being displayed as colored boxes. A *play line* slides across the track section as the video is being played, marking the exact frame position on the time-aligned annotation board. The play line can be dragged, too, for fast yet fine-grained video positioning. Time marks (seconds) in the horizontal top bar give temporal orientation. At all times, only one track is *active* (highlighted). In this active track, if the play line is located on an element, this element is the currently selected element (also highlighted). Its contents are displayed in detail in the upper right *track window*. The main window with menu bar and a text area for trace information is located upper left.

Coding is done on the annotation board (lower window) by marking the start of an element with the play line and then, marking the end. A track-specific edit window will appear and ask for the relevant attributes. Each attribute's input method depends on the respective value type: Strings are entered in a string input field, Booleans in a check box and ValueSets with an option menu containing the user-specified values. For MultiLinks, the user clicks a button to enter the *link markup mode*: A new window W appears to keep track of the list of chosen elements and the annotation board is free for user navigation to mark on/off those elements to be added/removed from the list. Clicking OK in W transfers the list of pointers to the currently edited element.

Each element can also be provided with a free-form comment, that is not part of the annotation scheme, for spontaneous research notes. Thus coded, the elements are displayed on the annotation board as boxes. Those attributes chosen for display (Section 4) have their values written into the boxes. On top of this, it is possible to *color-code* exactly one attribute (of type ValueSet). The user needs to specify one color for each possible value beforehand. To find the various research comments one has left in elements, those elements with a non-empty comment field are marked with a little square in one corner.

For coding and navigation, Anvil offers different zoom levels, context menus, keyboard shortcuts and bookmarks. Bookmarks are useful for marking places in the annotation for later reference. They can be added, accessed and removed through the main menu and are marked by small triangles in the time bar (see Fig. 3).

4. Customization and Configuration

How do you make a specific annotation scheme work with Anvil? Schemes should define layers of annotation and specify what kind of entities can be annotated per track. All of which must then be transferred to a *specification file* using XML syntax. Display options and documentation are also stored here.

For track organisation Anvil allows hierarchical structuring by providing *group* nodes. Group nodes can contain track nodes and other group nodes. The user can specify attributes for group nodes which will be *inherited* by the node's offspring. The group itself will not be used in annotation but displayed on the annotation board (see group nodes `posture` and `gesture` in Fig. 3). It is thus a purely organisational/conceptual entity.

The track/group hierarchy is specified by nesting the respective XML tags. Attribute name and type specifications are bracketed by respective XML tags. Display options are also inserted here. The attribute the user wants to take for color-coding is stated as `color-attr` in the track tag, attributes to be displayed in element boxes are marked by `display="true"` in the respective attribute tag. Here's an example:

```
<group name="gesture">
    <track name="phase" color-attr="hand">
        <attribute name="type" valuetype="phaseType"
                display="true"/>
        <attribute name="hand" valuetype="handType" />
    </track>
    <track name="phrase">
        <attribute name="category" valuetype="categoryType"
                display="true"/>
        <attribute name="correlate" valuetype="MultiLink" />
    </track>
</group>
```

[5] Groups are explained in Section 4

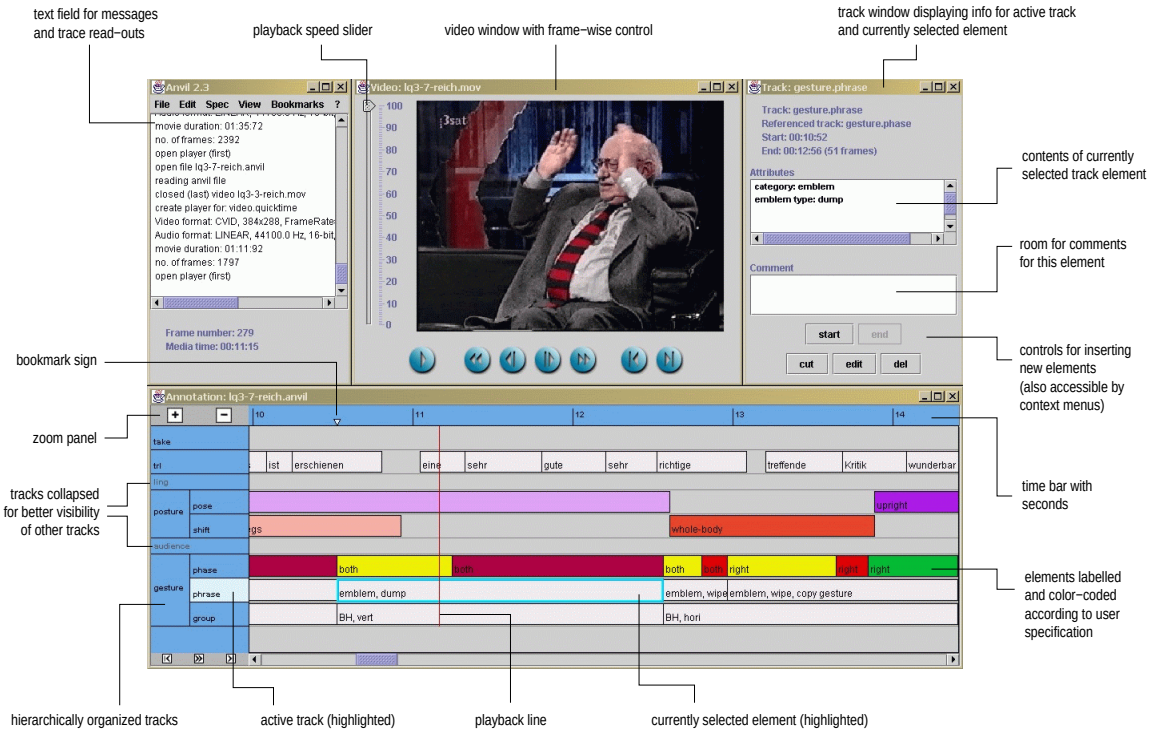

Figure 3: *Anvil's graphical user interface.*

In the sample (corresponding largely to Fig. 1), the three value types "phaseType", "handType" and "categoryType" are all user-defined value sets, specified like this:

```
<valueset name="handType">
    <value-el color="lightRed">both</value-el>
    <value-el color="lightBlue">left</value-el>
    <value-el color="yellow">right</value-el>
</valueset>
```

Note the definition of color values for the case that a track needs them for color-coding (as is the case for track `gesture.phase`[6]).

5. Project Management

This section takes on a broader, a *management* view on annotation and presents functions to connect Anvil to outer resources (import/export). For coding practice, the manual generation facility is introduced that should help increase consistency/reliability of coding.

For project management is it necessary to look at more than one annotation file. Therefore, the *project manager* allows the user to assemble file lists to so-called *projects* and store them for further usage, e.g. for text table export intended for global statistical analysis (see Section 5.2).

5.1. Import

For import, Anvil offers two functions. As Anvil is *not* suited for speech transcription/labeling, which needs very fine control of plackback and time marking, it offers to import transcriptions made with PRAAT[7]. Speech labels can be transferred to a selected primary track where they are converted to track elements. Labels can be words, syllables or whole phrases.

Another import function reads Rhetorical Structure, a hierarchical markup of text organization (cf. [10] for Rhetorical Structure Theory) that can be coded with the RSTtool[8] and imported to an Anvil (secondary) track in a flattened form (only rhetorical segments plus relation name and relation direction, i.e. forward/backward).

5.2. Export

Anvil provides two export functions. First, the export of a track as plain text. Thus, you can export the speech transcription to a text file for RST encoding and later on import the RST markup.

Second and more importantly, Anvil offers the extraction of the same track from a number of files (a *project*) to a single *text table* that can be used for statistical analysis in standard software packages like SPSS. The table comprises all track elements in lines. Rows contain the element's (1) start/end time and (2) all attribute values. In SPSS, such a table is easily imported, so that elaborate quantitative analysis can ensue.

5.3. Coding Manual Generation

As a quite innovative feature, Anvil can generate a coding manual from documentation of the specification file (inspired by *Javadoc*). The idea being that a coding manual must have exactly the same structure as the specification file, i.e. you give documentation of each track, each attribute, each attribute value. If a user inserts this kind of documentation into the spec-

[6] Names of tracks further down a track/group hierarchy are structured like a file path, using dots as separators.

[7] by Boersma/Weenik: http://www.fon.hum.uva.nl/praat

[8] visit http://www.sil.org/linguistics/rst/micktool.htm

ification file (bracketed with <doc>...</doc> tags), Anvil will produce a number of HTML pages, ready to use online by means of an internet browser. The look and feel is slightly reminiscent of Javadoc and mirrors the hierarchical track organization specified by the user. Its intention is to help coders keep a consistent view on their work for themselves (intra-coder) and in-between themselves (inter-coder).

Future work would consider how to insert examples and video stills in a semi-automatic way into the manual.

6. Related Work

Annotation on multiple layers is common practice. The BAS Partitur format [4] is an early example of a multiple-layer format where all entities (e.g. dialogue acts) point to a so-called canonical layer (word level). Partitur is a comfortable *file format* storing annotated entities in single lines and turns in single files each. Support through proper tools is only sparsely provided [5]. Also, it seems slightly outdated as its storage uses plain text and annotation entities are mere *labels* (i.e. strings) as opposed to *objects* with attribute-value pairs.

All more recent next generation annotation projects are Java-based, use XML for file exchange and have an object-oriented design:

The ATLAS project [6] deals with all questions of annotation and is theoretically based on the idea of *annotation graphs* [7] (reminiscent of *word hypothesis lattices* known from speech recognition) where nodes are times points and edges are annotation labels (e.g. words). Edges spanning several nodes represent higher level entities (e.g. dialogue acts or syntactic phrases). The ATLAS framework offers a logical view on annotations in form of an accessible API to ensure extensibility by other researchers. Also, a query language for selective extraction has been devised. The project looks highly promising but is at an early stage in terms of implementation.

EUDICO [8] is an effort to allow multi-user annotation of a centrally located corpus via a web-based interface. When finished the tool is to allow multi-modal video annotation. At the moment only a viewer is accessible. EUDICO is based on an earlier tool called MediaTagger which is in use at various research institutes but requires special hardware/software setup.

MATE [9] is an annotation workbench that allows highly generic specification via *stylesheets* that determine look and functionality of the user's tool implementation. Stylesheets also offer powerful retrieval mechanisms for annotated entities. Speed and stability of the tool are both still problematic for real annotation. Also, the highly generic approach increases the initial effort to set up the tool since you basically have to write your own tool using the MATE stylesheet language.

To conclude, Anvil compares to related work in that it is less ambitious: less complete than ATLAS (theoretical framework, retrieval/analysis functions), less generic than MATE (stylesheets), less multi-user/web-based than EUDICO. On the other hand, it is running, easy to install and continually being improved based on insights from practical annotation.

7. Conclusion

Anvil, a generic tool for annotation of multimodal discourse, was presented. It has a generic multi-layered approach and allows annotation of elements with attribute-value pairs, including cross-level pointers to other elements. Anvil is platform-independent, fast and comes with an intuitive, easy-to-use interface. It should therefore be of use to researchers dealing with multimodal discourse, for exploratory studies as well as for annotation of large corpora. Analyzing mass data is supported by an export feature that makes encoded data accessible for SPSS and other statistics toolkits.

The core functionality of Anvil is quite complete and running, the system easy to install and robust. Still, at the rim, there is much room for future work. Specification of tracks, attribute values etc. must still be done in XML which, although quite easy to do, should be managed by an editor. Furthermore, the project manager which has only recently been added needs some elaboration (insertion of more information like coders, date etc.). Also, the export function to text tables must be extended: you want *related* elements (related by temporal overlap or cross-level linkage) to appear in the same line for checking, e.g., correlation hypotheses. All this will be addressed in the near future.

8. References

[1] Klaus R. Scherer and Paul Ekman, *Handbook of Methods in Nonverbal Behavior Research*, Cambridge University Press, Cambridge, 1982.

[2] Elisabeth André, Thomas Rist, Susanne van Mulken, Martin Klesen, and Stephan Baldes, "The Automated Design of Believable Dialogues for Animated Presentation Teams," in *Embodied Conversational Agents*, Justine Cassell, Joseph Sullivan, Scott Prevost, and Elisabeth Churchill, Eds., pp. 220–255. MIT Press, Cambridge, MA, 2000.

[3] Justine Cassell, Joseph Sullivan, Scott Prevost, and Elisabeth Churchill, *Embodied Conversational Agents*, MIT Press, Cambridge, MA, 2000.

[4] F. Schiel, S. Burger, A. Geumann, and K. Weilhammer, "The Partitur Format at BAS," in *Proceedings of the First International Conference on Language Resources and Evaluation*, Granada, Spain, 1998.

[5] Jan Alexandersson, Ralf Engel, Michael Kipp, Stephan Koch, Uwe Küssner, Norbert Reithinger, and Manfred Stede, "Modeling Negotiation Dialogs," in *Verbmobil: Foundations of Speech-to-Speech Translation*, W. Wahlster, Ed., pp. 441–451. Springer, Berlin, 2000.

[6] Steven Bird, David Day, John Garofolo, John Henderson, Christophe Laprun, and Mark Liberman, "ATLAS: A Flexible and Extensible Architecture for Linguistic Annotation," in *Proceedings of the Second International Conference on Language Resources and Evaluation*, 2000, pp. 1699–1706.

[7] Steven Bird and Mark Liberman, "A Formal Framework for Linguistic Annotation," *Speech Communication*, 2000.

[8] H. Brugman, A. Russel, D. Broeder, and P. Wittenburg, "EUDICO. Annotation and Exploitation of Multi Media Corpora," in *Proceedings of LREC 2000 Workshop*, 2000.

[9] Laila Dybkjær and Niels Ole Bernsen, "The MATE Workbench," in *Proc. of the 2nd International Conference on Language Resources and Evaluation*, 2000.

[10] William C. Mann and Sandra A. Thompson, "Rhetorical Structure Theory: Toward a functional theory of text organization," *Text*, vol. 8, no. 3, pp. 243–281, 1988.