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

Deliverable status	Restricted
Contractual date of delivery	1 October 2001
Actual date of delivery	
Deliverable number	D1.1
Deliverable title	Software Specification and Workplan
Type	Report
Status & version	Draft version 2.2
Number of pages	37
WP contributing to the deliverable	WP1
WP / Task responsible	Jean Carletta, University of Edinburgh
Author(s)	Jean Carletta, Niels Ole Bernsen, Niels Cadée, Laila Dybkjær, Stefan Evert, Ulrich Heid, Amy Isard, Mykola Kolodnytsky, Christoph Lauer, Wolfgang Lezius, Lucas Noldus, Norbert Reithinger, and Andreas Vögele
EC Project Officer	Brian Macklin
Keywords	Tools specification, functionality, interface, architecture and platform, natural interactivity and multimodality data, annotation support
Abstract (for dissemination)	This report specifies the software to be developed within the NITE project, which aims to provide flexible, user-definable support for coding multimodal corpora. Software development will proceed in two stages. The first stage will involve extending the functionality of three different software packages (the MATE workbench, ANVIL, and The Observer Video-Pro) so that, taken together as a proof of concept, the three extensions can be seen to show how user needs in this area can be supported. The second stage will involve both integrating developments in MATE and ANVIL in order to provide some free support via public license software, plus further development of The Observer Video-Pro to better support the multimodal user community within a commercial framework. In addition to specifying software development, this report describes how the chosen approach is the best way to satisfy user needs given the support available from current packages.

NITE Deliverable D1.1

Software specification and workplan

17 December 2001
DRAFT version 2.2

This document gives an overview of our software specification and workplan. It is restricted to project partners. For more technical details, it refers to documents on a project internal website used by the project developers. This website is accessible to all project partners. Other readers may wish to visit this website as part of the review process. For this purpose, the username for this website is NITE and the password, w3NITEw3.

Authors

Jean Carletta[1], Niels Ole Bernsen[2], Niels Cadée [3], Laila Dybkjær[2], Stefan Evert[4], Ulrich Heid[4], Amy Isard[1], Mykola Kolodnytsky[2], Christoph Lauer[5], Wolfgang Lezius[4], Lucas Noldus [3], Norbert Reithinger[5], and Andreas Vögele[4]

1: HCRC, Edinburgh, UK. 2: NISLab, Odense University, Denmark. 3: Noldus Information Technology bv, Wageningen, The Netherlands.

4: IMS, Stuttgart University, Germany. 5: DFKI, Saarbrücken, Germany.

Section responsibilities

Section 1:	Introduction	Jean Carletta
Section 2:	Who needs tools for coding speech and video?	Jean Carletta
Section 2.2:	Driving Use Case	Ole Bernsen and Jean Carletta
Section 3:	What the user community already has	Jean Carletta
Section 3.1:	MATE	Amy Isard
Section 3.2:	Anvil	Christoph Lauer and Norbert Reithinger
Section 3.3:	Sonogram	Christoph Lauer and Norbert Reithinger
Section 3.3:	The Observer	Niels Cadée and Lucas Noldus
Section 4:	Specification	Jean Carletta
Section 4.1:	Strand One (using Anvil to explore new multimodal features)	Christoph Lauer and Norbert Reithinger
Section 4.2:	Strand Two (an open-source multimedia coding package)	Ole Bernsen, Jean Carletta, Laila Dybkjær, Stefan Evert, Ulrich Heid, Amy Isard, Mykola Kolodnytsky, Wolfgang Lezius, and Andreas Vögele
Section 4.3:	Strand Three (adaptation of The Observer)	Niels Cadée and Lucas Noldus
Section 5:	Points of contact among the strands	Jean Carletta
Section 6:	Relationship to other efforts	Jean Carletta
Section 7:	Workplan	Jean Carletta

Responsibility pertains to written form; most contents arose out of project meetings held in Odense, Denmark, 26-27 July 2001; Edinburgh, Scotland, 22-23 October 2001; and Pisa, Italy, 26-27 November 2001. All authors have approved the final deliverable.

Contents

1 Introduction

This report specifies the software to be developed within the NITE project. NITE is intended to improve upon current support for working with corpora of recorded audio and visual data from human-human and human-system interaction. This includes the transcription, annotation, coding, and analysis of this data, where the interaction can be purely spoken or involve other communicative modalities, such as gesture. ISLE deliverable D11.2 (Dybkjær et al., 2001) describes current software provision in this area and argues that the main shortcomings fall in three areas: support for the process of adding annotation and structured coding to a corpus according to a defined scheme, where "structured" means that tags refer not just to timespans but can relate to each other; support for the management of projects, especially the storage of metadata to encourage data reuse and information about the meaning and source of structured coding; and support for the development and analysis of new forms of structured coding. For these software functions, users would of course like to have software that is stable, covers the complete range of their needs, is easy to use, and can be adapted to new tasks. This NITE software specification is designed to concentrate on exactly these requirements.

Section 2 discusses and delimits the scope of the planned developments in terms of the intended users and functionality (Section 2.1), together with a driving use case which by itself requires all of the technological developments which we have identified as being needed by the user community (Section 2.2). It also points out the major considerations that have led to our choice of specification. Because there are a number of existing tools that go part of the way towards providing the required functionality, starting from scratch on one tool to serve all purposes would be counter-productive. Section 3 describes our general approach to serving this user community. This approach involves two stages of work: (1) development of three existing software tools so that they together demonstrate the required functionality, and then (2) integration of developments on the separate tools, to the degree that integration is possible. Section 4 describes stage 1, with separate sections describing planned developments on The Observer (Section 4.1), Anvil (Section 4.2), and MATE (Section 4.3). In each case development is described in terms of the tool's existing functionality, what they lack in service of this user community, and then what work is planned. Taken together, the developments on the three tools address the user needs identified in section 2.2. Section 5 then describes stage 2, our plans for integration of the functionality demonstrated in stage 1. These plans involve two strands of work: the integration of developments on Anvil and MATE to result in a NITE workbench — free, public license software which will both show our vision for software provision in this area and begin to supply that provision, within the constraints of what is possible outside a commercial setting — and the implementation of as many of the functions required by this user community as is commercially viable within the platform of The Observer. Section 6 ties together the specification by explaining why this two-pronged approach gives the project its best chance of fully serving the need for tools in this area.

2 Who needs tools for coding speech and video?

The tools that form the focus of the NITE project are intended to allow the user to describe interactive human behaviour. There are many conditions that affect how people interact with each other. Dialogues are different from small group discussions, which are different yet again from large ones; people behave differently when dealing with children than with adults; people who know each other behave less formally than those who don't; people from different

languages and cultures can have very different ways of interacting which are worthy of study both individually and in their cross-cultural combinations. In addition, people increasingly interact with computers, where the behaviours vary greatly depending on the interface of the system being used. Human interaction in all of these guises is of both academic interest and economic and social importance. In this section we describe the sectors that would benefit from improved tools for coding speech and video and outline in a general way the requirements of users engaged in these sectors. These sectors build on research from the fields of linguistics, psychology, anthropology, and human factors, among others.

2.1 Target sectors and typical uses

We expect NITE technology to be of interest to industrial sectors that can benefit from understanding how humans interact with each other and with machines. The industrial sectors at which NITE technology is targeted are spoken dialogue systems and multimodal human-computer interfaces, animation, communication technologies, and language documentation.

A **spoken dialogue system** is a computer program that can provide information to the human user via a spoken interface. The eventual goal for developers of such systems is to provide technology that can converse with humans in an entirely natural way, so that users need not adjust their interactional behaviours to each system's idiosyncracies. The difficulty in reaching this goal has been both understanding how human dialogue works and developing accurate enough speech recognition in dialogue contexts where the user's utterance is not highly constrained. If the system can obtain access to a video signal of the human user, one way out of this difficulty is **audiovisual speech recognition**, in which features of the video signal (primarily the position of the lips, but also general facial expression and posture) are used to help determine what the user is saying. The same information is even more critical for specialist **language learning software**, in which users learning a foreign language have their pronunciation corrected by the system. For these applications, NITE users will have collected and transcribed examples of arbitrary human speech (for lip position), human-human dialogue (for more general features in natural interaction), and human-computer dialogue (for specific features to train on in current systems) in different languages and dialects, and will wish to use NITE technology to mark up observable features of the video, transcription and speech signal, and then to look for correspondences. They may also wish to code some features of the video and speech signals automatically for correlation with hand-marked features, and to hand-correct the automatic coding.

More generally, NITE technology is useful in the development of **multimodal human-computer interfaces**. For instance, such interfaces might combine speech input (or a simpler command line interface) with deictic gestures, captured via touchscreens or video. Here one of the main issues is how one synchronizes events in the different modalities. For instance, when pointing to a location on an interactive tourist map, the user might say "I want to find a restaurant in this area", but the pointing gesture may lag or precede the corresponding speech, and there may be other hand movements in the same timeframe which need to be eliminated from the interpretation as distractors. With the ever-decreasing price and size of cameras, and theoretical work on calculating parallax so that trajectories can be interpreted correctly independently of camera angle, video is expected to become increasingly important for these interfaces, with suggestions that cameras can be placed wherever an interface happens to be (at automated teller machines, in cars, and so on). Although the information to be coded is different, the technological needs of these users are the same as those for spoken dialogue systems, with, for some applications, the possible incorporation of touchscreen data.

Sometimes there is a need to understand human behaviour outside of the context of human-computer interaction. A prime example of this is **lie detection**, for which the current physiology-based techniques are rather inaccurate. There are many rules of thumb about properties observable in the speech and video signals when a speaker is lying, amenable to analysis with this technology. Another example is that of improving current **communication technologies**. **Videoconferencing** makes it very difficult to hold properly interactive small group discussions because restricted visual information and audio lag interfere with natural turn-taking. One possible way to improve the next generation of systems is for them to recognize attempts to take the floor and adjust their camera angles and microphone settings accordingly. Similar information about the course of a discussion, when combined with speech skimming technology, would be useful for **video browsing**. In addition, in **multimedia desktop conferencing**, a better understanding of how people use shared applications such as whiteboards and web browsers, and especially, how they point within them, will lead to better systems.

In all of these sectors, the emphasis is on the automatic interpretation of human behaviour. However, wherever there is a need to understand human behaviour, there is also a need to synthesize it for more realistic **animation** and the creation of **artificial agents**. Because so many applications of animation rely on **talking heads**, there is currently much research being done on lip-synching, including how different facial expressions might affect the correct lip shape for some phonemes. Another topic of interest is turn-taking behaviours within dialogue, so that talking heads can be made to interact with the people watching them. Animation is important in a variety of industrial sectors, the largest of which is the production of cartoons and movies. However, as the web becomes more populated, engendering higher competition, it is also expected to be important for some web page front ends (for instance, as an alternative interface to search engines). This use of NITE technology employs not just the same functions as spoken dialogue systems work, but many of the same structured coding schemes. However, because of the flatness of synthesized speech and faces, there is more of an emphasis on understanding the observable correlates of emotional state for animation than for the other sectors.

Finally, simply documenting different languages and different kinds of interaction, including their gestures, is a goal of some potential users. Quite often this documentation is aimed at minority or endangered languages, and for some languages, there is an immediate need for action. The goals of these users tend to be social, but there are some economic benefits from acquiring this knowledge about the larger languages, mostly within **software localization** and to improve communication in the global economy, particularly within **cross-cultural business negotations**. Here there may be cultural constraints about the storage and re-use of data, perhaps placing heavier emphasis on project management than in other applications.

Although these sectors and uses showcase the relationship between video signal and speech, and thereby exercise the core of the new NITE technology, there are also large numbers of users who need something akin to NITE but without the video capabilities. This is the user community which needs the functionality which was developed in MATE, but delivered in a more mature package. Many of these users come from the areas listed above, but work with audio-only technologies (for instance, developers of spoken dialogue systems without a video connection and of teleconferencing systems). More surprisingly, it also includes users who are uninterested in dialogue or transcription, and therefore were not originally targeted by MATE, but simply need to mark up structured codings on written texts. This need is most prominent in service companies relying on the internet (for applications such as email filtering and the generation and analysis of web pages), but also exists more generally within, for instance, the publishing industry. For instance, Anvil has been used to mark up/annotate texts for a multi-

layered analysis with a view to terminology extraction (Vintar & Kipp, 2001). Although the core video technologies being developed on NITE are of less interest to these sectors, one side effect of the NITE project will be maturation of the MATE concept, benefiting these users.

Although we have been able to name a number of industrial sectors that show the commercial importance of developing the NITE technology, our main purpose is to facilitate basic research into how humans interact. NITE will be developing radically new tools for observing human language and behaviour, and especially the relationship between behaviours in different modalities during communicative activity. In the history of science, new tools often lead to breakthroughs in human understanding. One benefit of tools of this kind is that they allow data to be shared across the scientific community.

2.2 Driving use case

For the purposes of specifying software development, we have considered one use case in detail. As the reader can verify by cross-checking against section 2.1, this use case is intended to exercise most of the required functionality for the NITE technology.[1] For this purpose we have chosen the question of how the desire to take the discourse initiative within a small group discussion is signalled in speech, facial expression, gesture, and bodily posture. Understanding discourse initiative is important for many of our target commercial sectors, including animation, so that realistic discussants can be simulated, videoconferencing, so that camera angles and microphone settings can be adjusted automatically, and automatic analysis of meeting content. Although researchers have associated some behaviours with attempts to take initiative (changes in bodily posture and performing certain gestures (Beattie, 1985); backchannelling and heightened pitch (Oreström, 1983); looking at the current speaker), no one has definitively determined the full set of behaviours, nor determined how they combine. Research addressing this question would be implemented as follows.

2.2.1 Data collection

The research team would first collect audio and video recordings of a number of small group discussions for analysis. To collect information as rich as required, they would probably use one camera and one microphone per participant, setting them up carefully to maximize the information on each channel whilst minimizing interference. It is usual to also use an omnidirectional microphone and wide-angle camera to capture as much of the interaction as possible on one stream, both for analysis and as a rudimentary backup in case of equipment failure. At the same time as making the recordings, the research team would assign a unique identifier to each session, collect information about the participants (such as, perhaps, their age, sex, native language, dialect, social roles, degree of mutual acquaintance and educational background, and anything else thought to be important for interpreting behaviour) and describe the conditions under which the session occurred (such as the number of participants and the type of discussion). In addition, they would document the permissions obtained from the participants regarding use of the data. This kind of information is known collectively as "meta-data"; (Broeder, Offenga, & Wittenburg, 2001) gives a more complete discussion of the meta-data expected for corpora of this type.

Once the small group discussions have been recorded and catalogued in this way, they would most likely wish to render the recordings as computer-accessible audio and video files, for ease of future storage and processing. (The alternative is using specialist hardware and software that can access the audio and video from tape.) Because this research question may

[1] This use case, however, does omit one major requirement: it does not demonstrate the need for phonetic transcription. This requirement is important for other possible uses.

require the exploration of cause-and-effect relationships between behaviours from different participants which may be separated temporally as well as be present in different modalities, the researchers would probably choose to render each video signal as a different file, so that it is possible to view the cause in one window at the same time as the effect in a different one. In simpler analyses, it may be more sensible to combine the camera inputs into one time-synchronized frame showing all of the pictures. Besides the temporal restriction, this has the disadvantage of degrading the picture quality overall. However, it does make processing less computationally expensive.

2.2.2 Transcription

The next step would be to transcribe the small group discussions orthographically, according to some predefined conventions. This transcription would include time-alignment to signal at the level of individual words so that it is possible to relate what is being said to what is happening on the other signals, and to allow for the easy identification of areas of overlapped speech (since these are often areas where discourse initiative is either being passed from one person to another or is under question). Such transcription is useful just as a representation of the discussions when the researchers are exploring the data set. It also forms a useful basis for annotating and coding the data. Since transcription is time-consuming, researchers are likely to hire a range of temporary workers for this task, necessitating the use of project management techniques which allow them to assign recordings to individuals, track their progress, and indicate when transcriptions have been checked and corrected so that they are satisfactory. The researchers will also need to be able to document the transcription conventions that have been used in a way that makes this documentation easy for data users to find.

2.2.3 Coding on Individual Channels

Next, the researchers would be likely to hand-code a number of linguistic and behavioural phenomena on each of the individual channels thought to be related to the taking of discourse initiative. This might include pitch changes, utterance function (dialogue moves) and backchannel activity on the speech channels, and gesture, gaze direction, facial expression, and posture on the video channels. Note that coding some of these phenomena may require access to clear representations of the speech signal. The researchers will need the same project management tools as for transcription to manage this coding, including documentation of the coding distinctions being made. Coding further requires the ability to store two or more independent codings of the same phenomena and test how well they match each other, so that the reliability of the coding can be assessed. More important for coding than for transcription is storing the date on which the coding is performed, since one common reliability test is that of stability, or the degree to which the same person makes the same coding choices at different times. Since coding is time-consuming, it should be possible for work on the different channels to proceed independently and at the same time.

In parallel to adding hand-coding, the researchers might wish to add other codings to the individual channels using automatic processing. A typical example of this would be part of speech tags and syntactic information, where such processing is already reasonably accurate. (Syntax may be relevant to this research question, for instance, because it has been suggested that backchannels occurring at syntactic boundaries are qualitatively different from those that interrupt the syntax of another speaker.) Another possibility is to annotate utterance function automatically, and then hand-correct the results. Since automatic processes usually require specialist data formats, this step often requires the researchers to export the orthographic

transcription, re-importing the results of processing plus documentation of what processing has been done and what the coding means.

Transcribers and coders may both wish to add free-form annotations, or comments, to the data set. Such comments are common where transcription or coding is uncertain, or where phenomena occur which are interesting but are not picked up by any of the codings being employed.

Employing a set of codings entails a design of the study, as well as a design of the data format in which the codings and annotations will be stored. As well as storing the data and the description of what the codes mean, the data set should include a description of this data format sufficient for specialist import and export procedures to be written. Documenting the design of the study is also important for archive purposes, since without it, it may be difficult to understand the choices that were made.

2.2.4 Cross-channel Coding

Once individual channels have been coded, the researchers will wish to add codings that refer to events on different channels. These are likely to tie together the existing codes rather than to refer directly to the base transcription. Examples of such coding include identifying the turn structure of the small group discussion (since not all utterances will have been successful at obtaining the attention of the group) and combining utterances according to their discourse function into dialogue "games". Since the focus here is on determining how people seize the initiative in discourse, it may also include less well-defined clustering of all the behaviours that the researcher thinks are all related to one initiative grabbing event, so that these clusters can be found later on and analysed. As with individual channel codings, the coders may wish to generate annotations as well as formal codes, and will need to document the meaning of the information they have added to the data, as well as giving project management information. Part of this process may be carried out semi-automatically, by searching for all co-occurrences of certain phenomena from different modalities, provisionally labelling them with ad-hoc notes, manually verifying the nature of these co-occurences and then classifying (some of) them in view of the underlying research question.

2.2.5 Data exploration and analysis

After all of the data has been coded according to the study design, the first thing the researchers wish to do is usually to look at the data. This is more difficult than it sounds when there are so many different kinds of coding; showing them all at once according to some predetermined format overloads the human analyst. Instead, what the user requires is the ability to specify different ways of displaying parts of the data, concentrating on a few types of coding at a time. Cross-channel codings and automatic indexing may be added during this phase of a study. Although this phase of a study is quite informal, it is important for understanding the data well enough to design a formal analysis. The key here, in the face of such rich coding, is easy configurability of the data display.

As well as simply looking through the data, researchers will wish to firm up their intuitions using a range of techniques. The first is the generation of descriptive statistics. These may be simple counts and graphs, but of quite complex phenomena, identified using the same kind of querying as for indexing. The second is hypothesis testing using inferential statistics. The third is data modelling using a variety of techniques such as lag-sequential modelling or Markov modelling.

2.2.6 Data archiving

Subject to data protection constraints and participant consent, a major goal for producers of multimodal data sets is often to make them available for others to use. For data sets produced under the auspices of public bodies, this is often a requirement of funding. Archiving the data requires ensuring that the steps in data production have been documented as suggested, and sometimes converting the data set to another data format.

This use case outlines the steps required to address a rather complicated research question, and as such demonstrates the range of user needs for this technology. However, there is already existing technology for some of these needs. In NITE, we will be concentrating on providing new technology where support is currently thinnest, but showing how this support can be integrated with existing packages. The new support required arises from the complexity of the codings required for this kind of data set, and relates to coding, user-tailorable data display, indexing, search, and data extraction. Before proceeding to our plans, we first address a number of general issues affecting our choices.

2.3 The primacy of flexibility and structure over speed

One point which is worth making is that although there are a number of coding schemes for facial expression, gesture, syntax, dialogue structure, and so on, which are in widespread use (cf. WP2), every coding-based project is unique in some way. Coding schemes are improved over time and often must be tailored to the circumstances of the data being coded; new coding schemes emerge to match new theories or to describe phenomena not dealt with before. Even where well-established coding schemes are applied to a data set, they may be applied in new combinations, which require new display methods and analytical techniques. It is even controversial exactly what metadata should be associated with a data set, and whether or not entry of that metadata should be enforced. This suggests that in any support provided, NITE must be flexible enough to allow tailoring for the particular circumstances of the project. It is not enough to build a range of interfaces for known coding schemes; instead, what must be built is a general-purpose engine that can be configured for any reasonable coding scheme. The technological challenge is to allow this flexibility without making the software too complicated to use, or too liberal, with no constraints. In particular, developing the data structures inherent in the set of codings present on a single data set and defining mappings between these structures and possible data displays both require a degree of familiarity with design principles beyond the understanding of users accustomed to unstructured codings or to using whatever coding scheme is already supported by existing software. This design task is always present for structured coding projects, but is hidden within the task of providing special-purpose, dedicated software. If NITE is able to support this design task well, it will facilitate the production of a wide range of coding schemes, data displays, and coding interfaces, bringing benefits not just to the designer (since configuring existing software is faster and more robust than programming from scratch) but to the end user working with what the designer has built.

Immediately obvious from the detailed use case is that the coding structures which must be represented can be arbitrarily complex. Although most current coding schemes are largely tree-structured (for instance, syntax trees), the relationships among different coding schemes are not. It will be necessary to cluster codes from different schemes (for instance, to gather together all of the behaviours associated with an instance of taking the discourse initiative). The tags to be clustered can be arbitrarily far from each other in time (so that, for instance, a dialogue participant might backchannel for quite some time before finally gaining the floor) and relate to different data channels (e.g., one participant backchanneling, which leaves a trace on his audio stream, but then causes the current speaker to look towards him, leaving a

trace on his video stream). The structure of codings is essential to their interpretation for this kind of data.

One unfortunate side effect of the requirement for flexibility, and especially of the need to represent rich structures within the data coding, is that except in special circumstances, applications working with this kind of data are unable to stream the data. This makes processing inherently slow compared to working on tree structures or flat sets of tags. Speed of processing is often thought to be of utmost importance when working with linguistic corpora, largely because most applications involve the automatic processing of very large text corpora. Because the primary task that NITE will serve is the provision of interfaces for human coding and displays for exploring the data, speed is not as important here. Of course, NITE technology must be fast enough to make usable interfaces, but this requirement is an order of magnitude less onerous than the constraints for automatic text processing. In addition, the amount of material which must be loaded into memory at one time for human viewing or annotation is not so big as to cause massive problems. We estimate that a one-hour video recording, richly annotated, would use a maximum of 400 MB of RAM if the user needed the entire recording loaded at the same time. However, most users will have less rich annotations; one typical current data set uses 100 MB of RAM per hour recording. Where large scale automatic processing is needed on NITE data sets, so that speed and amount of material are issues, it is likely to need less rich structure than is present in the complete data set. In these cases, subsets of the data can be extracted and rendered as flat or tree-structured tags for further processing and re-importation, circumventing the inherent limitation.

2.4 Platform requirements

Another consideration to keep in mind when assessing user needs is the platforms that they use. Although many of our intended users are Windows-based, others, particularly those within academia and in industrial research laboratories, are accustomed to Unix workstations, or, more recently, Linux running on PCs. However, to state a requirement simply for cross-platform development or for parallel development on the two platforms would be to miss some of the most important ramifications of this platform divide. Traditionally, Windows users in this area are less technologically capable than Unix/Linux ones, and unable to tailor software to their purposes. It is more common to pay for software on the Windows platform, and for that payment to get support (such as training, a helpdesk, maintenance, and, in extreme cases, software customizations).

Although there are some paid software packages for Unix, software tends to be distributed for free, including the source code. Despite the existence of emulators such as Star Office, many in the Unix community have never encountered even the most common Windows-based programs (Microsoft Word, Excel) and are therefore unaccustomed to the interface techniques that they employ. Instead of monolithic interfaces, users are comfortable dealing with a set of smaller, single-function tools that together serve their needs, possibly with the addition of a few custom-written scripts. This means, for instance, that proprietary data formats, rather than being the key to continuing market share, reduce software uptake. It is difficult to plan for software support and maintenance when software is free. This is not to say that such programs are not supported, since tool users often make improvements and distribute them. However, it is fair to say that in this culture, users get by with a lower level of documentation and support than is expected in the commercial world.

Our assessment is that it is important to support users of multimedia data in both cultures. Members of the free software community are our technological drivers; they present the hardest technological demands, spurring developers to better functionality. As capable

programmers who are able to work around software that does not entirely match their needs, as well as to extend software where the source code is available, they provide the best chance of serving a wide range of needs after the project ends. Meanwhile, less able users need what functionality we can provide now and present the hardest demands for documentation and maintenance. Our workplan is specifically designed to benefit from this platform split rather than see it as a hindrance. It has two strands. The first strand culminates in free software that demonstrates the full flexibility users require, but which will be more suitable for those with some technical ability, particularly as maintenance will be outside our immediate control once the project ends and it is released to the community. The other strand aims to provide as much of the functionality that the community requires as possible within the scope of stable, supported, commercial software.

3 What the user community already has

We can see from the detailed use case that a wide range of functions is required. Users must be able to create video and audio recordings of interaction; transcribe them, orthographically and phonetically, including time-aligning the transcription to the signal; invent and employ multiple kinds of structured coding working either from transcription or from another representation of the speech such as a waveform, including analysing the reliability of the codings; apply automatic coding processes to the data, and hand-correct the results; display the data in different ways, as they explore the relationships contained within it; extract and index arbitrary parts of the data; build statistical descriptions of the data; and analyze the data using inferential statistics.

Whilst all this is done, users must be aided in managing their data — expressing metadata about the conditions under which the recordings were made, the transcription conventions employed and, since transcription of speech is never really complete, its level of reliability; describing the codings present on the data, who or what performed the codings, and, during the coding process, which recordings have been coded — both for their own purposes and so that the data can be reused by others. Although providing this functionality may seem an enormous task, there are already tools providing some of it, as argued in ISLE D11.2 (Dybkjær et al. 2001). Thus NITE will concentrate on how best to provide the missing functions.

The best way to characterize existing support is that the core functionality of hand-coding transcriptions, displaying codings, indexing, and extracting data are not yet provided, but the peripheral functions are. Transcription is supported via packages such as Transcriber; some users employ xwaves for this purpose, and others use standard word processing, sometimes aligning words with signal after the fact in xwaves. Both descriptive and inferential statistics are well supported in SPSS and Excel, as long as it is possible to export the data to be analysed to them in some kind of simple tabular format. Common kinds of automatic coding, such as part-of-speech tagging, already exist. It would be time-consuming to reimplement these functionalities. Instead, we intend to make it possible to link our software with existing tools by having them share a common data format: XML. Many tools, such as Transcriber, Microsoft Word, and some part-of-speech taggers, are already starting to employ XML as a native data format or to include export options. The filters for extracting data from XML into tabular formats are quite simple, as are those for uptranslating text-based transcriptions.

This leaves the core functionality of hand-coding, including project management, the coding of structured information which links existing codes in complex ways, configurable data display, indexing based on queries matching structural and temporal constraints on the data, and extracting subsets of the data based on the same sorts of queries. There are three existing

tools that address this core functionality, but each fail to provide the full support needed. They are MATE, Anvil, and The Observer.

3.1 MATE

The MATE workbench is a software tool for the display and annotation of XML encoded speech or text copora. The workbench allows a user to display and edit existing corpora, add new levels of annotation, perform queries over part or all of a corpus, and display or output the results. The format of the display and editing actions are set up using rule-based stylesheets based on a pre-standard version of the XSLT transformation language. The workbench provides a number of pre-defined stylesheets for use with particular annotation schemes, but its major strength lies in the fact that the stylesheet language is sufficiently high-level for writing stylesheets to be significantly easier than writing an editor from scratch. This means that users can use one of the pre-defined stylesheets if it meets their needs, or write a new stylesheet of their own (possibly based on one of the existing ones) tailored to their particular data and annotation needs. Queries can also be performed to select a subset of the corpus for further processing within the workbench or for output to external tools. The workbench has in-built support for standoff annotation in which annotations are not all stored in one document but are linked by means of pointers. This allows the editing of one level of annotation without disturbing other levels and also makes it possible to have multiple annotation tiers with overlapping branches. The MATE workbench has been used successfully on several corpora for coding phenomena such as dialogue structure, tutoring strategy, and the use of metonymy, but there are a number of areas in which improvement is necessary.

The support for using the raw speech data while annotating is very limited — it is possible to play a section of speech, but it is only possible to display a waveform of the entire speech file, and the waveform display is not properly integrated with the rest of the workbench. There is also no support for spectrograms. The MATE workbench has no video capability at all, and will need to be extended to permit the annotation of video resources. As is common with the first implementation of a radically new idea, the workbench is under-documented and not as robust as would be required for widespread use. Neither the query language nor the display objects have been optimised, and it is necessary to have a detailed knowledge of the complex (and mostly undocumented) interactions between them in order to write an efficient stylesheet. The language in which graphical user interface actions are specified needs to be both simplified and extended, and it must be possible to load more XML material into the workbench at one time. The workbench is also too slow in practice for many common coding tasks.

The MATE workbench is freely available and the code can be downloaded under a GNU public licence from http://mate.nis.sdu.dk/.

3.2 Anvil

Anvil is a research tool for the analysis of digitized audiovisual data. It allows the user to code human behavior and other visually accessible information in temporal alignment with speech and other auditory signals. Anvil was written as part of a PhD project on nonverbal communication at the University of the Saarland with support from the Deutsche Forschungsgemeinschaft (DFG) and builds on experiences with mass corpus annotation of dialogue acts within the speech-to-speech machine translation project *Verbmobil*. It has been actively used to encode video samples of a German TV show with nonverbal communication events, mostly gestures, and linguistic information.

In Anvil, for each type of behaviour (for instance, hand gesture or bodily posture) the user first defines an *annotation scheme* representing the range of behaviours that can occur. Then the user can use the software to divide the video into *behavioural units*, each represented by a code, for each type of behaviour. During coding and subsequent display of the data, codes are shown in layers, with one layer for each annotation scheme. These layers are displayed one below the other, running from left to right, just as a musical score (or "Partitur", in German) shows instrumental parts in parallel. Behavioural units are depicted as boxes (rectangles) whose left and right borders correspond to their start and end points on a common time axis, the width thus being the duration of the element. Adding labels and colours to these bars allows intuitive comprehension of such a behavioural "Partitur" where temporal relationships between the layers, categories and durations can be captured at a glance. Anvil aims to make coding as intuitive and fast as possible, and to give the one most informative view of the resulting data. Although for the most part one can consider coding for the different types of behaviour to be independent even though they are temporally synchronized, it is possible to link behavioural units across the annotation layers into more complex structures. Anvil's control and data files are all in XML, the W3C standard markup language. This means that Anvil users can exploit the many tools that exist for the manipulation and transformation of XML files, especially when they wish to carry out data analysis.

Anvil is written completely in Java, making use of a recent extension called Java Media Framework (JMF). JMF is part of Sun's optional packages for Java2 and comes with a common Java API. JMF enables audio, video and other time-based media to be added to Java applications and applets. This package, which can capture, playback, stream and transcode multiple media formats, extends the multimedia capabilities on the Java2 platform, and gives multimedia developers a powerful toolkit to develop scalable, cross-platform technology. Because Anvil is written in Java, it is in theory platform-independent. In practice, the JMF part is only guaranteed to run on Windows, Linux and Solaris. It is not clear whether or not it will work on Macintoshes. The Anvil system was developed on a Pentium III (500 MHz) with 256 MB RAM, and successfully tested on Windows (98, NT, 2000) and LINUX platforms. Since it is written in Java2 with JMF 2.1.1 it should run on any platform that has those components installed. Anvil works with the common video formats that are supported within JMF, such as AVI and QuickTime. The URL http://www.dfki.de/~kipp/anvil gives more information about the exact formats and codecs supported.[2] Anvil is currently restricted in the formats it can support by the underlying support provided in JMF. However, JMF is continually being expanded to cover more formats and codecs.

Anvil has some very attractive features, such as its Partitur-based visual representation, which make it an attractive choice of tool on which to base further development. However, it is a very basic software package and needs further work before it will properly support users. Currently, for instance, it lacks on-line help, keyboard shortcuts for performing annotation, and the ability to add free structure annotations or "comments" to the data, even though all of these things are important during video coding. Some users may have multiple video tracks, but Anvil is limited to displaying one. Anvil is also currently intended to support only the video coding process and no related functions. Users must directly edit an XML file in order to configure the annotation schemes to be applied. It has no search, import, or export capabilities; instead, the user is expected to work directly with the XML which Anvil reads and produces. Although Anvil allows some linking across annotation layers, the basic

[2] A video file *format* defines the framework for data, metadata, and header information, such as how many audiotracks the file contains; a *codec* defines the compression format for the video and audio in the data part of the file.

expectation is of independence, with the output XML conforming to one rigid, non-tailorable structure.

3.3 Sonogram

Linguists and phoneticians need to see various representations of the speech signal in order to, for instance, transcribe phonemes or annotate prosodic features. Anvil and Mate already allow for simple spectrum visualization in terms of speech amplitude over time. However, other visualizations are often needed. One existing tool which produces them is Sonogram, (http://www.dfki.de/~clauer), written at DFKI by Christoph Lauer.

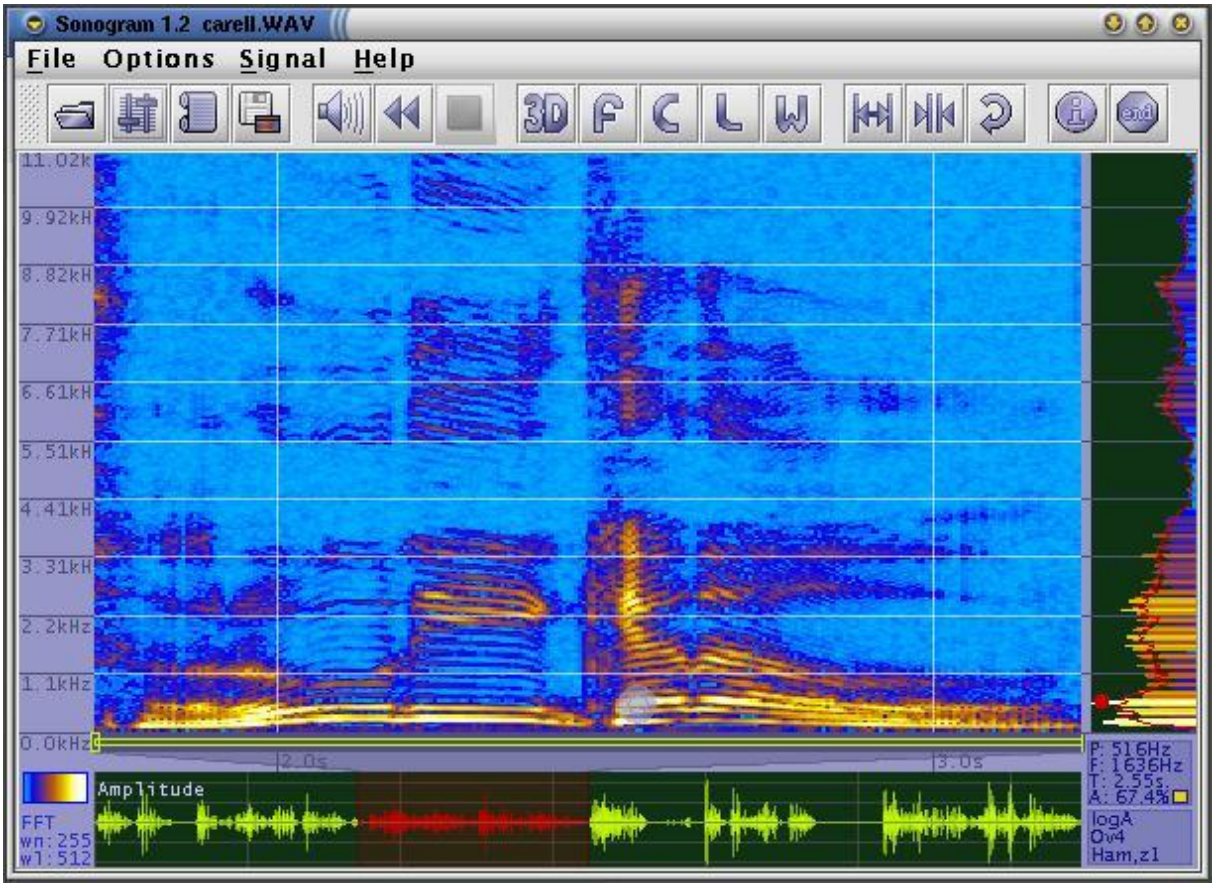


Figure 1: Screenshot of Sonogram.

Sonogram is a highly sophisticated, configurable tool for visualizing spectrums. For a screenshot, see figure 1. Sonogram contains a wealth of signal processing algorithms that the user can adapt and tune to exactly fit his needs. For example, it can generate spectrograms, logarithmic frequency views, two or three dimensional surface plots, and 'Linear Prediction Coefficients' (LPC), used primarily in speech recognition. Sonogram is implemented in Java2 using the 'Swing' library, and can work with data in any multimedia format supported by Java Media Framework (JMF). As well as working with audio files, JMF allows Sonogram to open the audio tracks of video files. 3D surface plots are only available when Java3D, a free toolset that implements the Silicon Graphics OpenGL and the DirectX interfaces in Java, is installed. Both JMF and Java3D are toolsets orginally developed by Sun and ported to Linux (http://www.blackdown.org/) by the open source community.

3.4 The Observer

The Observer (Noldus, Trienes, Hendriksen, Jansen, & Jansen, 2000) is a professional program for coding and analysis of video data (stored on tape or in digital media files) of any kind of behavioural process. It is a commercial package that works on Windows platforms (Win 98, NT, 2000). Installation is easy and straightforward. You can design configurations for coding, and make a library of configurations for re-use. However, the richly structured and inter-linked coding schemes typical for the NITE project are not supported in the current version (version 3).

You can code from the keyboard in real-time after some practice. You can have an overview window with all the codes in your configuration displayed while you code. The event log, which is a table with time running vertical, keeps track of the time and all different behavioural classes (or tracks). The event log is linked to the video time, which allows for quick searching and reviewing. However, it takes quite some time to learn to master these options. A horizontal timeline like in the ANVIL program might be a more intuitive display of multiple tracks. Text transcription is possible, but text is treated as free comments on time intervals. For NITE, more structure is needed for speech annotation, for example specifying words within phrases, and phrases within sentences. It would be nice to be able to do video annotation, graphically marking up a specific part of the video image where a certain behaviour occurs.

The Observer data files contain the date and time of scoring, and every code is stored with a time stamp. The names of the people who did the coding can be stored as independent variables. There is a need for more elaborate project administration, containing for example, which MPEG files still need to be encoded, and which annotations have been checked for accuracy. XML import/export is not currently supported.

The Observer has several modules for data analysis. Most of these require some training and practice before they can be used. Below we list each module with its basic functionality as well as needs for enhancement as identified by the NITE partners, and what is currently under development as a result.

- The time-event table gives a basic table-like overview of the annotation.
- The time-event plot looks like the ANVIL horizontal timeline display format. However, there is no display of spoken text, sound waveforms or other signals, and it is not possible to create links between different tracks.
- Reliability analysis lets you compare coding of the same video data by two different people. However, you cannot compare more than two people's coding simultaneously. The Kappa statistic and confusion matrices will be completed in the next few months (before the end of 2001).
- The elementary statistics module allows you to get frequencies and distributions for all codes. It is not possible to display graphical plots. Currently the output file name is limited to four characters, but this will be increased in the next version.
- Lag sequential analysis can be used to analyze the frequency of transitions between behaviours. Probabilities of transitions will be available in the next version.

In conclusion, the interface of The Observer needs more attention. As it is now, the program is not designed to do basic tasks in a simple way, and there is a counter-intuitive way to set up the video display. Speech annotation and analysis have not yet been implemented. Support for complex coding schemes, with many cross-linked tracks that can be linked and grouped, is needed. In addition, adding XML import/export functionality would enable communication with text transcription programs and other linguistic annotation tools.

4 Specification

Our set of user needs is rather complicated. On the one hand, we want to support end users properly. This means working towards software that is properly supported and maintained in the longer term, something that is difficult to arrange in the context of a European project. On the other hand, we want to make sure that the support we provide is sufficiently flexible to meet the users' true needs. Too often, research is guided by the tools available, rather than the other way around. It would be an inappropriate use of our funding simply to develop software that requires no technological advancement and for which industry could already make a solid business case.

As we have described, The Observer can already be used for some types of work on multimodal corpora, and supports a wide range of user functions, but does not handle the sort of highly structured annotation common in linguistically motivated work and does not provide a mechanism for importing or exporting data in XML, the format that is becoming a *de facto* standard for corpus annotations. Anvil provides a simpler XML-compliant version of the core coding functionalities present in The Observer, and through a limited capability to link codes in different annotation tiers begins to support structured annotation, even though it does not allow the user to give a formal definition of what that structure is. It uses a single uniform visualization of coded data, but one that is perhaps more acceptable to users in this community than the one provided by The Observer. Because it is relatively simple compared to the Observer, and is implemented using the Java Media Framework, it is arguably quicker to adapt in new development, especially for the addition of speech signal visualizations as are needed by some parts of the community. Finally, MATE shows how best to support structured annotation fully, including formal definition of what the structure is and flexible definition of visual display and coding interfaces.

As a result, our plans are as follows. In the first strand of the project, we will use Anvil as a quick testbed for the new audio and video capabilities that we require. This strategy will allow us to test the novel features that we think users require much more quickly than would be possible otherwise. In the second strand, we will develop an open source workbench that supports the core functions still required by the community. This will include both a uniform visual interface that, based on the annotation tier concept, will ignore any further structure present in the data, and a flexible command language engine for supporting structured annotation, using the processing model developed in MATE. Finally, in the third strand we will migrate The Observer to serve this community better, incorporating as many of the changes which we have identified as can be supported in their business case.

Note that even though our work in strand two is aimed at the typical user of free software, the majority of whom we expect to be on the Unix/Linux platform, since development will be done in Java, nothing precludes the results being used under Windows. We do not expect to be forced to make development choices that tie us to any particular platform. Our experiences on MATE suggest that Java code developed on one platform does run well on other platforms, with the proviso that each platform requires separate installation procedures. Despite this, it is our assessment that for many users, especially Windows-based ones, only software provided with a commercial level of support, such as the software resulting from strand three, is likely to be fully successful.

4.1 Strand One (using Anvil to explore new multimodal features)

Although we can use Anvil within this project, we may not re-use Anvil code within our own development process. Our approach is to write a series of plug-ins which will work with either Anvil or other annotation software which handle some of the new multimodal features

which the NITE user community requires. This approach will allow us to test these plug-ins early in the development process using Anvil, transferring the plug-ins, which we will own, to the NITE workbench. The main functionality which will be demonstrated by developing Anvil is the linking of annotation to representations of the speech signal, the specification of coding design, and the production of more detailed video coding using graphical overlays on the video stream. Software development will involve the following tasks:

1. Fusion with spectrum visualization using Sonograms. Like Anvil, Sonogram requires some adaptation before it can be used by Anvil and the NITE workbench. Sonogram takes a speech signal plus offsets from the beginning of the file specifying the start and end times for the part of the signal to be analysed. Anvil and NITE should be able to call Sonogram in three ways. First, at start-up, Anvil and NITE should be able to ask Sonogram to generate a display, given an audio file name and start and end time offsets. Second, given a representation of a time interval on an Anvil or NITE display, it should be possible to place a button on the user interface to display that time span using Sonogram. Finally, when Sonogram is running, it should be possible to end its run.

   The simplest way to implement this functionality is to allow Anvil and NITE to pass messages to Sonogram using TCP/IP sockets or http ports. These techniques can exchange the data as textual or XML information. Alternatively, it may be possible to integrate the tools at the Java level, perhaps using Java beans. For the Anvil development work, we will first implement a solution that loads Sonogram into Anvil so that Anvil can call Sonogram. This step must be taken after Anvil is restructured as a plug-in.

   When we integrate Sonogram into Anvil, we do not intend to find a way to display spectrograms as one track of the Anvil display, as this would be quite difficult to incorporate into the present software. However, we will try to synchronize the timelines of the annotation tracks and of Sonogram so that they move in tandem.
2. Graphical specification editor. It is not easy to add tracks in the specification files, or generate completely new specification files without specific knowledge of XML. A Visual XML specification file editor would be more intuitive for this aim. A first solution is currently under development by the original author of Anvil. We will provide the experience with this editor for the further development of project management tools for the NITE workbench.
3. Graphical video markup. The second signal-oriented add-on to Anvil that we will implement is the ability to directly mark gestures in the video stream. As an example, it should be possible to encircle the arm of a person during one gesture phase and to link it to the annotation in the gesture annotation track in Anvil.

   With Java Media Framework most common multimedia features are quick to implement and easy to use. As a technical solution, we are currently considering the following. The simplest way to play any multimedia file is to generate a class named '*Player*' which has no configuration possibility. The '*Processor*' class which extends the '*Player*' class has a lot more adjustment possibilities and provides the flexible functions we need as standard '*Player*'. With the class '*Effect*' new classes can be plugged between internal modules of this '*Processor*'. Using this framework, each pixel of each frame of a video is accessible. This nice feature in JMF makes it easy to implement filters, and also magnifications and other effects. One interesting effect is to brighten up some specific regions in a video. A disadvantage of this solution is that it increases the processor and memory load significantly. However, future improvements in processor speed and the java virtual machine will solve this problem.

   An open problem is, as for the integration of Sonogram, the time aligned replay of the annotation in the video and in the textual mark-up track. We will try to use the same solution we envisage for Sonogram to solve this synchronisation problem.

   In that way graphical mark-up is easy to implement. (Processor Class has one disadvantage to '*Player*' Class which is that '*Player*' take less memory and CPU resources as '*Processor*'.)

Although Anvil does *not* provide convenient facilities to transcribe speech or other phonetic data like intonation, we do not intend to provide these. As discussion in section two, there are

already good tools around which support transcription, making this functionality outwith the scope of this project.

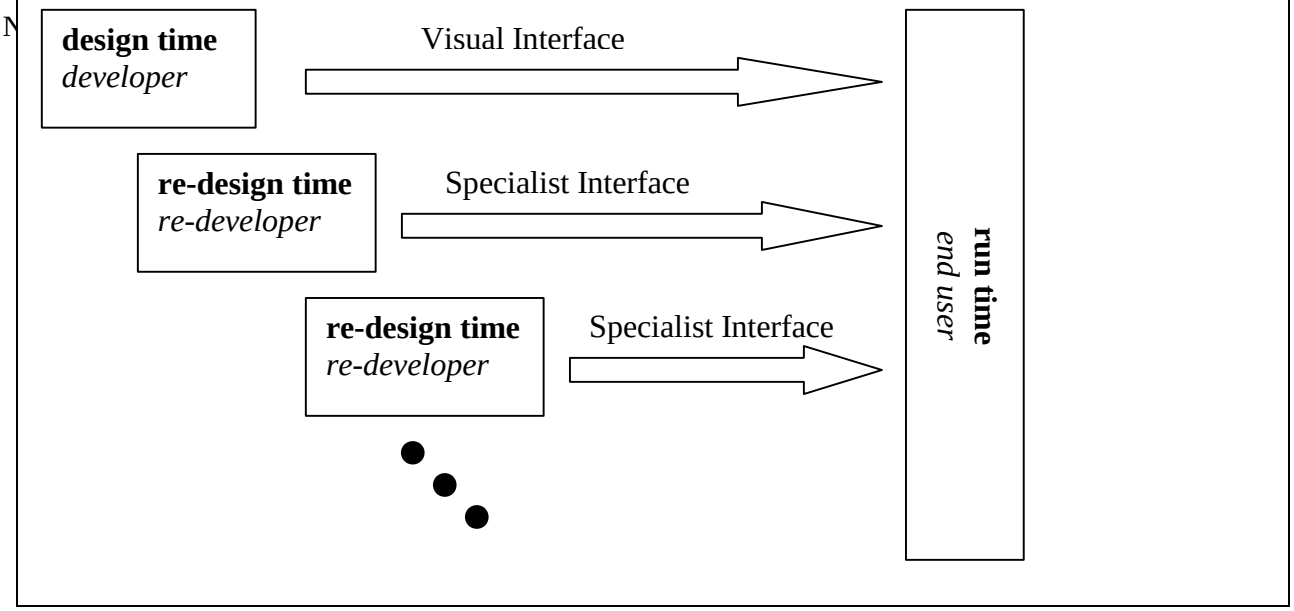
4.2 Strand Two (an open source multimedia coding package)

In the three existing packages, we can see that there is an essential tension in coding support. As in Anvil and The Observer, support is relatively easy to provide when codes are simply timestamps on the underlying signal, organized in tiers, which might have some hierarchical decomposition within them, but which are structurally independent of each other. Although Anvil allows some linking between codes in different tiers, there is little that the user can do to visualize or query these links, and each one is treated as ad hoc rather than as an example of some underlying regular structure. The structural assumptions of Anvil and The Observer make it possible to specify a data format and display that will work reasonably well for all data of this simple structure. On the other hand, MATE allows one to specify any complex coding structure one might like, but at the price of having to design both the data format and the data display. These tasks require designers with technical skills over and above those normally found in the intended end users. Both approaches are correct. Anvil and The Observer suffice for some needs, circumvent the need for design skills, and therefore make it quicker and easier to complete a coding task. However, when their structural assumptions are inappropriate — as they are for many codings with a linguistic orientation — the MATE approach makes coding feasible, and ensures that projects are not required to design and implement coding software from scratch, as was previously the case. The more structured the set of codings to be applied to a corpus, the more necessary it is to employ design skills specifically in service of that project. We need to keep both approaches in mind if we are to support user needs fully.

Given this understanding, our approach is as follows. We will provide two ways for end users to use the workbench. The first is through a standard, uniform visual interface. Once data has been imported into the workbench format, it will be possible for end users simply to start using this interface to code and view the data. This interface will concentrate on capabilities that are basic and universal to multimodal corpora. However, it will of necessity be restricted to representing a set of timestamped annotation tiers of the sort currently supported by The Observer and Anvil. The second is through tailored interfaces specified via a flexible command language interface based on stylesheets and the processing metaphors that arose in MATE. This second method will allow interfaces to be designed with a radically different feel from the uniform visual interface, and with the capability to represent complex data structures. End users are likely to use the visual interface when it has the functionality they need, employing re-developers to add tailored interfaces when it does not.

This approach, sensible as it is in light of user needs, does not fit into a standard software cycle of design and run time. Figure 2 describes the software cycle that we intend to use. The figure shows that writing specialized interfaces using the command language interface of the workbench is a re-design activity by which re-developers add more functionality to the workbench. Other functionality which lies outwith the scope of this project, such as the development of data import and export functions, can also be seen as re-design activity. It is important that the resulting software be evaluated in reference to this cycle. In particular, there are two facets to evaluating the command language interface: whether or not re-designers can write interfaces using it, and whether or not end users can use the interfaces that result.

Figure 2. The NITE workbench software development cycle.



4.2.1 Data format

Essential to our approach is the use of a unified data format that can be interpreted by the visual interface but is also suitable for more structured codings. Data format is not a very important question for codings that are simply streams of timestamps arranged in tiers. There are many possible and obvious data formats that will suffice. In cases where the program provides all of the functions the user requires, or has the capability of importing and exporting data to other programs, the user need know nothing about how codes are stored, so the exact choice is unimportant. However, users are likely to begin their projects by adopting the structural assumptions that make the visual interface appropriate, and then move towards structured codings as their research questions become more complex and as they become more comfortable with the coding enterprise. The structured approach will require them to extend the unstructured data format and to write corresponding data displays. For this reason, we need to use a well-supported and flexible data format for both tasks.

XML is the best choice for the basic data format. It is an intrinsic part of the way in which the MATE workbench works, because XML allows one to specify a unique structure for a data set formally. Together with a wide range of existing XML tools, this means that even for one-off designs, such data sets will already have a good deal of software support. Most current corpus material is produced in XML. XML's precursor, SGML, formed the basis of previous text encoding standards (such as the TEI and CES). XML is current being adopted widely by corpus initatives such as Talkbank, Atlas, and OLAC.

Like most data formats, a single XML document is hierarchically structured. This does raise some design difficulties for working with complex data structures. MATE's approach is to represent data using a set of documents whose inter-relationships are specified using links similar in form (but not semantics) to the hrefs that connect web pages together into a network. The practice of storing different annotation levels in separate files and linking each annotation level to a base level containing timing information has several benefits. Timings for the higher levels can be calculated from the base level, which means that there is no way that timings can get out of synchronization with one another. It is possible to add new annotation levels without affecting the existing ones in any way, and indeed people adding new levels do not need to know anything about the structure of the other annotations already in place. It is also possible using stand-off annotation to publish annotations on a corpus without publishing the corpus itself; this means that the annotator can purchase an XML corpus, add the annotations in XML hyperlink format, and then publish the results without infringing the copyright of the original corpus, but the new annotations will be usable by other researchers who also own a copy of the same corpus.

In order for this data format to work for the visual interface, the tier and timestamp ordering must be retrievable from the XML files. One way of ensuring this is by specifying as a general design principle that each XML document should represent a different tier (whether that is a single stream of timestamps, or a set of streams of timestamps which hierarchially decompose), with the XML tags ordered temporally within that document. (In data representations outwith the remit of the visual interface — for instance, a representation of a universe of entities as required by coreference annotation — not all data is linked to the timeline, but conforms to other kinds of orderings, also represented using the XML structure.) We can ground this principle out into the following design constraints:

1. Codes are represented as XML elements.
2. Where orthographic transcription exists, it is represented in a standard way. (Most simply, by using element contents for orthographic transcription and for no other purpose, but less strictly, it may be possible to allow a data importer to specify how to find the transcription. This constraint is necessary so that the visual interface knows how to display the transcription on the screen.)
3. Where codes are timestamped, their start and end times are represented in a standard way. (Most simply, by having two reserved attributes for storing the offset from the start of the recorded data for the start and end times, but less strictly, it may be possible to allow a data importer to instead specify some process, such as a more complex XML query or computation, by which these start and end times may be obtained.)
4. A code's timestamps must be organized so that the code starts no later than it ends. Timestamped codes can be either instantaneous or contain some time interval, but cannot have negative temporal extent.
5. The XML ordering of sibling elements in a document must be consistent with the temporal ordering represented in their timestamps. (That is, if tag A comes before tag B in the XML order, the end time of tag A must be less than or equal to the start time of tag B. We do not require that adjacent tags abut each other temporally, but they must not overlap.)
6. Hierarchical decomposition of elements within a single XML document describes hierarchical decomposition of an annotation tier.
7. Where a timestamped element hierarchically decomposes into a set of children given in the same XML document, temporal information given in the set of children must be consistent with temporal information given in the parent element. (That is, the start time of the first child in the XML order must be greater than or equal to the start time of the parent, and the end time of the last child in the XML order must be less than or equal to the end time of the parent.)
8. XML documents relating to the same data recording must represent timing information in comparable ways, so that whatever methods are used for calculating time offsets in two different documents, the temporal relationship between codes in the two documents is clear.

These constraints make it possible to build representations of the data against timelines without a human author further specifying what the XML file structure means. Note that constraints 5 and 7 do not imply that any particular annotation tier or hierarchical decomposition of a tier fully spans the data; in some coding schemes, there may be uncoded intervals interspersed throughout the data, even when coding is complete. Implicit in these design constraints is the fact that there is no implied timing relationship for linked elements other than that provided by their timestamping. We intend links to be used for representing purely structural information, such as the relationship between a deictic referring expression and the gesture that loosely accompanies it, or the relationship between two expressions that refer to the same entity. In XML, links may tie together elements either from the same document or from different documents. In this design, links specify a structural relationship

for the linked elements; their temporal relationship, if any, is given solely by the timestamping, and, if the elements are in the same document, by the hierarchical document structure.

In the early stages of NITE development, it may be necessary to further develop these constraints. The latest version of them will be kept at http://www.ltg.ed.ac.uk/NITE/data-format-NITE/data-format-NITE.doc.

4.2.2 Data import

Importing data into NITE means transforming it to XML that adheres to the design constraints expressed in the previous section, and, if we are to allow flexibility concerning how timestamps and orthography are represented, specifying the mappings inherent in constraints 1 and 2. We may also wish to make it possible to give more information about how to impose further temporal structure on the set of XML documents that represent a single dataset. For instance, in a hierarchical decomposition where the set of child elements abut each other temporally and the parent element exactly spans the set of children, it is common practice not to place explicit timing information on the parent. Instead, the timing of the parent element should be inherited from its children. This is useful because it makes it impossible for the timing information to become inconsistent. In will also be necessary in some cases to break our dictum that cross-document links do not specify temporal information. It is quite common in multiply-annotated linguistic corpora to have several trees point to the same base elements (for instance, dialogue moves, syntax, and disfluencies all pointing to the same orthographic words). In these cases, the words usually appear in a document separate from any of the other structures, but are intended to be related temporally to all of them. The easiest way to facilitate this sort of inheritance, whether it is within or across documents, would be to allow the data importer to designate XML elements for which timings should be derived from the (same- or different-document) children in this way. This information could most easily be stored using a reserved boolean attribute. All of this information would most naturally be stored in an XML file accompanying the data, but we may wish to consider methods for facilitating the creation of this file.

Although providing a visual interface is intended to make it possible for end users to start coding without having to learn anything about XML, stylesheets, or interface techniques, of course someone must know about XML in order to convert existing data so that it complies with the specified data format. Since existing data formats vary widely, data conversion techniques will be one-off processes for which it would be impossible to provide sufficient support. Writing routines that will import and export data to other formats (such as BAS-Partitur or Atlas Interchange Format) is beyond the scope of this project. However, the overall workbench interface may wish to make it possible to integrate routines providing such functionality into the workbench, so that users can push a button on the workbench to get data imported and exported, rather than run an external program. Thus data export routines should be seen as an addition to the workbench at re-design time, just as specialized interfaces are. Export should be easy to facilitate; since NITE data is in XML, the most natural choice of mechanism for transformation to other formats is a stylesheet, where the stylesheet produces tabular text or another type of XML rather than a graphical display. But we have stylesheet processing as part of our workbench architecture. The same argument holds for existing XML formats that do not comply with our format constraints. For processing which is not based on stylesheets, in principle, as long as the conversion routines are written in something callable from Java it should be possible to at least specify a process by which re-developers attach buttons to the workbench that perform the conversion.

4.2.3 Project management and the provision of metadata

One of our user requirements is for maintained information about the data sets accessible at a particular site, especially the state of any annotation present on the data or in progress. This requires the user to maintain information about, for instance, the types of annotation devised for a data set, how to find the documents in which the annotations are formally defined, whether the annotation can be performed using the standard visual interface, whether there are any special displays or interfaces relating to that annotation which have been built using the command language interface, who performed the annotation and when (so that there is an audit trail for project work and so that the reliability of annotations can be properly tested), and so on. Although this sort of information currently is not often properly maintained, it is invaluable for maximizing value of data sets which are heavily reused. As part of this project, we will specify, implement, and document an interface by which the information of this type for each data set associated with a particular instance of the workbench can be entered and maintained.

4.2.4 The visual interface

The NITE interface will have to be a quite complex one. It has to cater for raw data perception, transcription visualisation, informal and free-style comments, multi-level and cross-level annotation, annotation analysis, annotation comparison, search and inspection of search results, and perhaps more, including the trivia of enabling file saving, printing, deletion, overview, duplication, import and export. Moreover, it has to enable all the things mentioned in the context of facilitating the annotation and analysis of full natural interactive communicative behaviour. In trying to solve this large-scale problem, we initially aim to preserve the unity of the main individual functions of the interface, such as the unity of the annotation environment, the unity of the analysis environment, and the unity of the search environment. In other words, we would like the user to face as few different interface environments as possible. If this does not prove possible, we will regrettably have to further complicate the interface of the NITE tool.

Given the aim just stated, the following interface specification primarily focuses on the annotation and analysis interface(s).

Taking into account the structure of the user interface of the MATE, Anvil and Noldus software as well as current standards and user interface guidelines for software design, we suggest that the NITE workbench user interface should have the standard components of a main window and should enable three main types of user activity:

annotation and transcription of raw data (audio and video corpora);
information retrieval from annotated multi-modal corpora – queries and data analysis, search and visualisation of retrieved data, etc.;
coding schemes configuration – adding a new coding scheme from scratch or by modifying an existing one.

In order to provide the user with a uniform way of doing annotation and information retrieval for multi-level and cross-level annotation and annotation analysis, as well as to provide several ways to display annotated files, we suggest to organise the user interface in the following way.

The NITE user interface should consist of the following five main components:

the **main window** which contains the main menu, the title, etc.;
the main window **toolbar** which contains the changeable (contents-sensitive) set of buttons;
a changeable amount of ***panels*** of the *i*-th class of phenomena to be annotated – 1 up to 10 panels;

the **raw data windows** displaying the different types of raw data (video, audio);
the common control board.

In addition, numerous palettes (dialogue boxes) with several controls will be provided for the user to work with different coding schemes (to insert/delete tags), to visualise tags, and so on.

We suggest to implement the following scenario of annotation:

to select a class of phenomena to annotate using a particular coding scheme;
to *edit* (insert/delete) the marker of tags relative to the "copy" of the common time-line;
to *visualise* tags.

Markup of a "copy" of the common time-line (or string a text) is performed as a two-step process:

inserting the "*empty*" marker of a proper tag onto the time-line;
visualising the tags, having chosen an adequate style of visualisation (from the pre-defined set of possibilities).

This approach allows us to provide a uniform style of work with the annotation tool: for any level of annotation and any coding scheme, the user performs the same set of actions: choosing from the panel a class of phenomena or a coding scheme, choosing the appropriate button (the appropriate tag) from the coding palette, inserting the marker of the tag into the copy of the common time-line on the panel, and, finally, choosing the style of graphical visualisation of these tags on that time-line (on the annotated panel).

There are several coding schemes for each annotation level, but from a software development point of view all of these can be presented in a common way, namely as a table containing a hierarchical list of tags. For internal data representation and storage of such tables, the implementation of relational database technology can be useful.

The kind of user interface architecture described above will make it possible to satisfy the following requirements of "NITE use case UC-1" as well as ISLE Deliverable D11.1:

Annotation support at different levels, of different modalities.

Each of the different levels will be supported by a separate annotation (or transcription) *panel*. Each panel can be activated for annotation, shown or hidden.

Multi-level and multi-modality annotation.

It will be possible to display several panels activated and shown inside the NITE frame window at the same time (one below the other or overlaid).

Annotation support of interaction across levels and modalities.

The interaction across levels will be implemented as a function of the common frame application (*NITE's main window*) and the possibility to interact between (to process) several panels within one common application – the NITE workbench - rather than several autonomous ones (MATE, Anvil and Noldus software).

Common time-line for different levels.

The "common time-line" is actually the time line of the raw data. Each panel also contains the time-line which is related to the common time-line and, in some sense, duplicates it for all different levels in order to facilitate the annotation process.

Possibly synchronised view of different layers of annotation and of different modalities.

The common time-line has, of course, an internal representation (data structure) within the NITE workbench.

Having a common internal representation of the "time-lines" of different levels of annotation, it is possible to synchronise the views of different panels. The common **control board** (which is a part of the NITE frame window) allows the playing of events (the marked up text) along these time-lines.

Open and flexible architecture of software (and user interface).

It is easy to add an additional level of annotation. It will cause the appearance of a new panel of annotation (as well as a new palette with the appropriate coding scheme). Thus, it allows us to modify the functionality of the NITE workbench as well as to make an open software architecture.

Support for easy addition of new coding schemes (and for defining new visualisations).

In order to add a new coding scheme into the workbench, we have to provide the user with a new palette which will contain the new set of controls (tags).

Most importantly, perhaps, such a tool (or toolset) has to be robust, stable (and work in real time).

Having implemented part of the workbench for annotation at just one level (or coding scheme), we can easily spread out results to other levels thanks to the *uniform* approach described above. This should improve the robustness and stable work of the NITE workbench as a whole.

Separation of user interface from application logic (external representation) and data (internal representation) so that each can be changed independently.

The user interface design presented above is described at some abstract level which does not include the details (features) which depend on the internal data representation of the elements of the GUI. It allows us to change and modify both the internal data and the user interface independently.

The following is a, no doubt partial and certainly arbitrarily ordered, list of basic properties which must be present in the interface in order for the interface to serve the natural interactivity annotation process. In the present context, a 'property' is an abstract property which could be realised graphically in many different ways.

Raw data must be made perceptually accessible to the user at will. The user must be able to look at the video whenever necessary and switch it off at will. The user must be able to listen to the audio track whenever necessary and switch it off at will. The audio track and the video track must be controllable independently of one another. The same applies to viewing of other raw data, such as logfiles and graphical representations of acoustic information (the latter is considered raw data for present purposes even if this might be contended). Acoustic raw data, video raw data, and graphical representations of acoustic information should all come with a visible timeline. It must be possible to navigate back and forth in the data based on the timeline. It is not clear if the above has to apply to logfile information as well.

At least orthographic transcription of acoustic data must be available on the screen. Phonetic transcription remains an open issue. The orthographic transcription itself may be done elsewhere, for instance using Transcriber. Note, however, that any imports, such as from Transcriber, should conform to the interface conventions adopted for the interface being specified.

Orthographic transcription must be viewable in musical score format.

Appropriate facilities must be present for annotating any aspect of the video, including free-form comments on what goes on in the video as well as use of standard annotation schemes

for facial expression, emotion, gaze, gestures of all kinds, lip movements, bodily posture, actions, etc.

Annotation levels (including orthographic transcription) should visibly share a common timeline. This means that, however those annotation levels are being represented, the common timeline corresponding to the audio and/or video and/or graphical representation of acoustic events must visibly "run through" all annotations.

Up to 10 annotation levels must be shown simultaneously on the screen.

Given the fact that showing as many as 10 annotation levels may be a relatively rare occurrence, allowance could be made for screen extensions to make it possible, for instance through scrolling. However, a more standard number of annotation levels must be shown on the screen without scrolling.

It should be possible to link to the common timeline: points (no duration), timed entities of any length (sub-words, filled or unfilled pauses, words, phrases, acts, as well as larger communication units of any kind).

It should be possible to clearly label those entities during annotation.

Labelling of entities, if not done free-form, could be done by selecting from a palette elsewhere on the screen. The palette would contain all possible tags belonging to a particular coding scheme.

It should be possible to visibly link to the common timeline: long-range dependencies, such as co-references, cause-and-effect or event conditions.

It should be possible to visibly link different annotations, such as linking facial expression annotation to transcription, or linking gesture annotation to speech act annotation. This will enable researchers to establish "clusters" of communication phenomena.

It should be possible to remove individual links as well as all links between two different levels. (If all links are removed temporarily, this may be done by saving the file with the links before removing them).

It should be possible to experimentally name the cross-level links mentioned above. This means that the annotator will not necessarily have a palette-represented annotation scheme at hand during annotation. If other solutions turn out to be too costly, a simpler solution is to make it possible for the experimental annotator to open a free-style editor window in which to insert cross-level link names. If the experimental annotation yields promising results, the annotator could then proceed to create an experimental coding scheme-cum-palette.

It should be possible to open simple editing windows at any time. These windows may be used to insert additional observations related to the annotation process as well as to the analysis process.

It should be possible to re-order annotation representations on the screen. Re-ordering may affect cross-level links. Links should be preserved but a certain clutter may be unavoidable.

It should be possible to visibly compare different annotations of the same raw data/transcription. Due to screen space limitations, this might require level representation re-ordering and/or scrolling. It is not mandatory that annotation comparison actions can be done at the same time as annotation is possible. Annotation comparison actions might be doable only with the system in non-annotation mode.

Modes: a common top-screen menu line should provide access to all other system functionality. If it proves easier to do so, some or all selections of alternative system

functionality may put the system in non-annotation mode. Examples are: search, annotation comparison, possibly including simple statistics, etc. This means that it is possible to specify the different system modes one-by-one.

Palettes must be able to refer to (mark up) single-level phenomena as well as cross-level links.

Given the fact that the phenomena to be annotated in trying to understand natural interactivity may not only be quite complex but also temporally extended or even linked to an indeterminate extent, it must be possible to view transcriptions and annotations at different levels of resolution, from viewing few-seconds-duration cross-level, cross-modality annotations close-up to viewing minutes-long stretches of transcription and annotation birds-eye. This imposes additional requirements onto the representation of levels and links.

Work process: it is essential to make sure that the interface behaves appropriately at all times during the annotation and analysis processes. We should not expect serious users to annotate according to several annotation schemes simultaneously. This means that the structure of one palette/one annotation scheme - one set of annotation levels active for annotation would be a valid one. Note, however, that since we are after finding new regularities which may well be cross-level or cross-modality, it should be possible to link several annotation levels during annotation.

The URL http://www.ltg.ed.ac.uk/NITE/visual-interface-NITE/visual-interface-NITE.doc gives the latest version of the visual interface specification.

4.2.5 The command language interface

The command language interface will follow the processing model based on stylesheets that was demonstrated in the MATE workbench. The MATE workbench has a modular architecture that consists of an internal database representation of the data, a query language and processor, a stylesheet language and processor, and a display processor. The internal representation (IR) represents the structure of a set of hyperlinked XML files. There are functions for loading and outputting XML files into and out of the database. The query language and query processor are used to select parts of the corpus structure. The stylesheet language describes structural transformations and the stylesheet processor implements this language. The output of a transformation applied to a document can either be another document or a set of display objects. The display processor takes the display object output of a stylesheet transformation and shows it to the user. Where the stylesheet specifies how to capture user input and how to interpret it as changes to the underlying data, this mechanism produces not just a data visualization but a graphical coding interface.

The basic MATE concept worked, but the workbench itself was buggy, slow, and had an inadequate language for describing the treatment of user input. Our command language interface will be essentially a re-implementation of MATE. However, there are a number of important differences here. First, the XML world has moved on since MATE was developed. In particular, there is now a standard language for stylesheets (XSLT) that we can employ in the interface. This makes it easier for re-developers to use, since the type of user we envision using this interface is likely either already to know XSLT, or to find learning it useful for other purposes. Second, there are many more Java software libraries that we can draw on now. Swing, which will form the basis of our display and interface capabilities, is much better developed on. The Java Media Framework makes it simpler to develop audio and video capabilities. Rather than having to develop XML parsing and stylesheet execution ourselves, we can use existing software available from Xalan or Saxon. These developments will make our software faster to write, faster to execute, and less prone to bugs. Third, we intend to support the process of authoring information, essential to executing specialized interfaces,

about which stylesheets are associated with which data sets and what calls will run them. In MATE, this information took an idiosyncratic, non-XML and undocumented form.

The URL http://www.ltg.ed.ac.uk/NITE/CL-interface-NITE/CL-interface-NITE.doc gives the latest version of the command language interface specification.

4.2.6 Data querying, indexing, and extraction

Data querying should be implemented with a three-step process in mind. In the first step, the user will deal with the graphical window (the Query Window) which includes controls for building the query. In the second step, an expression in SQL (Simple Query Language) format is built as a result of Query Window processing. In the third step, the SQL expression can be run on a relational database (i.e. MS Access or MS FoxPro) or applied to the stored annotated files, thus completing the query. If the annotated file store is implemented with XML, the SQL expression has to be translated into the MATE query language and then applied to the stored annotated files.

There are five things a user might want to do with a query once he has written it. First, he might wish to store it for later use. Second, he might wish to paste it into a stylesheet that he is developing using the command language interface. Third, he might wish to see all matches to the query immediately highlighted in the visual interface. Fourth, he might want to use it to add indexing to the corpus material; depending on their permanance, the index codes can be seen as temporary markers for finding interesting parts of the data, or as additions to the corpus markup. Finally, he might want to use it to copy just the matching data into an extracted sub-corpus.

The URL http://www.ltg.ed.ac.uk/NITE/query-interface-NITE/query-interface-NITE.doc gives the latest version of the query language interface specification.

4.3 Strand Three (adaptation of The Observer)

Noldus Information Technology is currently completing The Observer 4. Version 4.0 will be released before the end of 2001, and version 4.1 will follow in the beginning of 2002. These releases will contain entirely redesigned project management, configuration design and analysis modules. The new design will increase the usability of the software, as compared to version 3 that was tested by the NITE participants (see section 3.3).

This section lists required functionality, with regard to the NITE project, for The Observer 5 (a tentative version number for the next major upgrade after 4.1). As The Observer is a generic tool for behavioral observation and analysis, Noldus Information Technology needs to weigh these requirements against the needs of several other user communities (such as psychologists, usability testers and movement scientists). The requirements listed below are a compilation of the priorities expressed by the NITE partners present in Odense (July 2001) and the priorities defined by the Noldus marketing department. Desired new functionality has been rated as high, medium and low priority. As far as NITE-specific features are concerned, the ranking represents the view of the NITE partners, not Noldus Information Technology. This means that a few items may require further discussion between the NITE partners and the Noldus R&D team (with Niels Cadée and Lucas Noldus serving as liaisons) before the list can be regarded as final. Ideally all items can be developed, but this is eventually dependent on available R&D capacity (at Noldus) and a more detailed cost-benefit analysis. As soon as the list of priorities for The Observer 5 has been finalized, development will start.

4.3.1 High priority

- **Improved usability –** Improve the user interface to make the program easier to use, especially for new users. For instance, add a "configuration wizard" that guides the user step-by-step through the design of a coding scheme.
- **Integrate all "modules" into one application –** Make the current observation module an integral part of the Observer application, so that all data and video related functions are always accessible, all file-related actions in one File menu, etc. The current division is artificial and arbitrary. This implies a redesign of the software architecture.
- **Link analysis and video** – Create a link between analysis results and video. For example, right-click a cell in the output table of reliability analysis, select Play Video in the context menu that pops up, and the corresponding video fragment is played. This will add a substantial amount of interactivity to the system.
- **Hierarchical structured configuration –** Support for tree-structured hierarchical configurations, or classification of codes. For example, "phrases" and "phases" of gestures (both are finite series of mutually exclusive elements; a phrase is always made up of a series of phases).
- **Replace proprietary file formats by industry standards –** The Observer 5 will be based on a database architecture, using a standard database (e.g. Access) with the possibility for users to import and export files in standard formats, e.g. ASCII text files or XML (see next two items). This database will replace the many different profiles, configuration files, customization files, project files, and data files that have evolved over the years.
- **Export data in XML -** Export the event log, with a separate XML file for each data stream (channel, track). This and the next three requirements are essential for the NITE user community to use The Observer. In this way it is possible to exchange annotations between The Observer and the XML based software that linguists use.
- **Export configuration in XML -** Export the configuration, or coding scheme, as a DTD (Document Type Definition) file.
- **Export project info in XML -** Export project information (also called metadata) with references to media files, data files, configuration, etc., as an XML header file.
- **Import data from XML -** Import individual channels or tracks into an existing project. Examples could be speech transcription created with Transcriber, or linguistic coding (like prosody) of transcribed text.
- **Record metadata -** Offer a framework to record all relevant metadata, i.e. data explaining the conditions under which the annotations were collected. These could be data on the persons doing the coding, the age and gender of the subjects, the language, and any information about specific tasks the subjects were carrying out. The location and status of files also needs to be recorded. The status could for example be 'to be annotated', 'annotated', or 'reviewed'.
- **Increased string length** – Allow 25 character for subjects, behavioral elements and modifiers. Literature research has shown that with 25 characters we can handle most typical names of behavioral elements.
- **Restyle video display window** – Redesign video window according to current UI standards. Add "dockable" control buttons and sliders.

4.3.2 Medium priority

- **Improved configuration checker** – This will make the program more user-friendly and facilitates the creation of correct coding schemes for inexperienced users. Add an option for real-time check (while making the configuration).

- **Support multiple video streams** – Add support for the synchronized playback of up to four video streams (AVI, MPEG-1, MPEG-2 or QuickTime movie files).
- **Timeline display -** Show the event log as a horizontal timeline display with parallel tracks, like in DFKI's Anvil program. This display should always be synchronized with the video stream. It may be useful to provide different display modes: the event log as present in The Observer 3 (a vertical table), a vertical table with separate columns for all tracks (comparable to the time-event table in The Observer 3), and a horizontal timeline display (comparable to Anvil and the time-event plot in The Observer 3). Users can then choose the display mode that fits best to the task at hand (data entry, retrieval, analysis, etc.).
- **More advanced queries -** Support more advanced queries than currently possible, especially with hierarchical levels. See Map Task for examples of queries (http://www.hcrc.ed.ac.uk/maptask).
- **Visualize transcription -** Visualize parts of the speech transcription. See Map Task (http://www.hcrc.ed.ac.uk/maptask) for examples of the way transcribed speech can be visualized.
- **Show names of codes in graphical plots of annotation -** Display annotation codes or speech transcription inside colored bars in the timeline display. When the text is too long, it should be truncated. The user should be able to switch this display on and off.
- **Include video editing functionality -** Add functionality for digital video editing in order to produce highlight compilations. Add simple overlay effects and options to add text and graphics between clips or as overlay. This should allow a user to quickly make a summary clip of, for example, all attention grabbing initiatives of a particular person in a group discussion.
- **Compliance with Windows GUI standards** – Add functions that users expect in a modern Windows package, such as Undo, Close workspace, Delete workspace, Rename. Use standard Windows functions to select items for copying, refresh etc. Offer standard shortcuts in event editor (for cut, copy, paste, insert, delete, etc.).

4.3.3 Low priority

- **Enforce recording of metadata -** Provide a function as in Microsoft PowerPoint where the properties sheet of a project pops up whenever changes are saved. This procedure will help to ensure that all relevant metadata for a project is entered. This will facilitate archiving and exchange of projects.
- **Show text in timeline display -** Display codes (e.g. behavioral elements) or speech transcription inside the colored bars in the horizontal timeline display. When the text is too long, it should be truncated.
- **Free-form annotation -** Comments are already supported for the vertical event log in The Observer 3. In the horizontal timeline display, comments should be shown by putting a small marker in the corner of the colored bars depicting coding elements. When the user moves the mouse over this marker, the comment will appear in a pop-up window.
- **Graphical markup of video –** This should allow the user to mark an area in the video image that deserves special attention, like a gesture or a specific facial expression. The marking should stay visible for the entire duration of the corresponding coding element.
- **Multiple comparison -** The Observer 4.1 will already contain the Kappa statistic and confusion matrices for pairwise comparison of annotations. The reliability analysis module should be expanded to do automatic analysis of repeated pairwise comparisons, with results displayed in a matrix.

- **Multi-user support** - Allow the use of the system by multiple concurrent users, for instance up to 4 concurrent annotators (who are entering data) and up to 10 concurrent viewers (who are reviewing and analyzing a corpus, playing video, etc.). This requires a new multi-user database architecture.
- **Customizable user interface** – Allow the user to switch between a "full" UI (for use by the expert) to a "restricted" UI (for use by a novice), where the “restricted” mode only shows a subset of the functionality. This will make the program easier to use for non-experts, e.g. students. At this stage it is not yet clear if different user profiles are necessary, and if so, how many, and what the feature sets of each profile should be.

The following two requirements will first be tried out by DFKI in Anvil. After that, we will learn from Anvil how to synchronize and display these data.

- **Display waveform -** Reserve one track in the timeline display for the waveform of the speech.
- **Display spectrogram -** Reserve one track in the timeline display for the spectrogram of the speech. This is useful especially for coding of prosody.

These requirements outline functionality that is in many cases essential to meet the needs of the user community that NITE addresses. Several requirements, like an intuitive horizontal timeline display, a link between analysis and video, a standard database, support for multiple video streams and multiple concurrent users, will also appeal to other user communities.

5 Points of contact among the strands

Although we have specified three different strands of development, and, within the second strand, separate visual and command language interfaces, there are points of contact among the strands in our development plan.

First, the visual and command language interfaces for strand two will share in common libraries of functions for their core capabilities. Both interfaces need a set of display objects that together form the repertoire of possible data display arrangement techniques. Some typical display objects are panes, horizontal and vertical lists, and textboxes. It would be inefficient for the visual interface to develop one set of display objects but the command language interface to use another. Instead, we will develop a shared library of display techniques that we can draw upon in the development of both interfaces. Similarly, the two interfaces will share a set of user interface primitives that can be used to obtain user input.

Second, rather than developing completely new software for working with video data in strand two, we will be re-using libraries of code arising from strand one in this work. Similarly, the work in strand one about developing a graphical user interface for designing annotation tier structures will feed into work in strand two on the visual interface.

Third, strand one allows the early testing of interface ideas. Developers in both of the other strands, but particularly in strand three, will be considering the evaluation of strand one results in forming more precise implementation plans. Although code can not be re-used between strand one and strand three, successful designs in strand one are likely to be directly re-implemented in strand three.

6 Relationship to other efforts

NITE is not the only project that is engaged in facilitating annotation of corpora. The Linguistic Data Consortium of the University of Pennsylvania, NIST, and MITRE are currently engaged in a set of projects with simiar intentions to NITE. Their approach centres

on the definition of data models appropriate to different kinds of annotation and aims to support annotation by providing library routines for (1) reading and writing XML data that fits a prescribed DTD corresponding to the data model and (2) data handling, including the addition and deletion of annotations, for the model. Using these library routines, developers can more easily build end user applications than if they need to develop this infrastructure from scratch. They are currently working on two different data models. The first, which they call the "annotation graph" (http://www.ldc.upenn.edu/AG/), views annotations as simple timespans on a single signal and is tailored for the needs of speech researchers. This data model is intended to support high-speed processing for large amounts of data. However, it does not allow for the expression of structural relationships among annotations. The second, which arises from the Atlas project (http://www.nist.gov/speech/atlas/), is intended to allow for some structural information, but at the time of writing, the exact affordances of the data model and intended user community are not yet clear.

NITE bears some similarity to these projects, but differs in three substantial ways. First, NITE is the only one of the projects to concentrate major work on the annotation of video signals. Video does form part of target functionality for American developments; for instance, the Open Language Archive Community (http://www.language-archives.org/), have expressed the need for video annotation. Annotation graphs apply equally well to video as to speech if only timestamped annotations are needed. However, as we have seen, the user requirements are more complex than this, requiring, as they do, the identification of regions of a video signal and the expression of structural relationships among video annotations and annotations about the underlying linguistic data. Because of NITE's focus, it is in a better position to serve these needs. Second, although the American developments concentrate on provision of library routines for software developers to draw upon, a major focus of the NITE project is the provision of end user interfaces. Third, although NITE aims to define a default data model that can be used for the most prevalent types of data and will underpin the standard visual interface, NITE also takes the approach that the most appropriate data model for working with any particular data set is the one that most closely matches the data design. This argument is made more fully in (Carletta, McKelvie, & Isard, 2002 (to appear)). Thus, even though the command language interface of NITE and the various American efforts are all intended to support the work of software developers, they do this in different ways. The American developments provide support for working with a static data model, but not for writing end user displays and interfaces. NITE provides support for the development of end user interfaces by supplying libraries of display and interface routines and a stylesheet language, already known to many developers, for assembling interfaces, but relies on software developers to understand the data model and the XML data representation behind the data set they wish to support.

Figure 3: Overview task list.

Task No.	Task description	Site responsible
1	**Strand One**	
1.1	Specification, implementation, and documentation of Sonogram plug-in; demonstration of use in Anvil.	DFKI
1.2	Specification, implementation, and documentation of plug-in for facilitating coding design according to the Anvil/NITE visual interface data model; demonstration of use in Anvil.	DFKI
1.3	Specification, implementation, and documentation of plug-in for graphical video markup; demonstration of use in Anvil.	DFKI
2	**Strand Two**	
2.1	Specification of data format design.	EDIN
2.2	Specification of the data model for NITE use of stand-off XML, definition of an API for working with the data internal representation, implementation, and documentation.	EDIN/IMS
2.3	Specification of shared display and interface libraries.	NIS/EDIN
2.4	Implementation and documentation of shared display and interface libraries.	NIS
2.5	Specification of query language; definition of API for working with query module and implementation and documentation of query module.	IMS
2.6	Specification, implementation, and documentation of extensions to XSLT for stylesheet language.	IMS/EDIN
2.7	Specification of data importation functionality to meet data model required by standard visual interface.	NIS
2.8	Implementation and documentation of data importation functionality to meet data model required by standard visual interface.	
2.9	Specification of project management functionality.	EDIN
2.10	Implementation and documentation of project management functionality.	
2.11	Specification, implementation, and documentation of overall interface including extensibility to include functionality defined by re-developers.	NIS
2.12	Specification, implementation, and documentation of visual interface.	NIS
2.13	Specification, implementation, and documentation of command language interface.	EDIN/IMS
2.14	Specification, implementation, and documentation of query language interface	NIS
2.15	Specification of data extraction and indexing capabilities.	IMS
2.16	Implementation and documentation of data extraction and indexing capabilities	

7 Workplan

The cross-site nature of development on strand two of the project, and the close relationship between strands one and two, means that we require close collaboration in order to complete

our work. Figure 3 gives an outline view of the WP3 development tasks for these strands of the project. We have organized our workplan around a number of internal discussion documents which give more technical detail than would be reasonable to put in this overview document. Each document is to be authored by one project partner (in one case, two project partners both of whom have expertise which is necessary for making decisions in the area), and then discussed by the development team before being finalized. In the workplan, discussion is scheduled over a period of at least 10 working days for each document, in order to ensure that adequate feedback from the project partners is received. These documents are hosted on the developer group website, http://www.ltg.ed.ac.uk/NITE/.

The timecourse of our workplan is as follows. There is a major milestone one year into the project. By this time, we expect strand one of the project to have completed making plug-ins that work in Anvil but can be transferred to the NITE workbench, relating to both the Sonogram and new video markup capabilities. It is possible that the coding design functionality may also be complete by this time. Strand two will have available a prototype of the standard visual interface based on rapid prototyping techniques and written in C++, as well as the design documents relating to the command language interface. We expect some implementation related to the command language interface to have been completed, but due to the way in which this work is intended to support software developers, we do not expect end user interfaces to arise from this sub-strand of work by this point in the project. Strand three will have incorporated the most urgent of the changes identified into a working version of The Observer. At this point in the project, end users will be able to evaluate the new strand one plug-ins using Anvil, the prototype visual interface, and the changed version of The Observer; developers will be able to evaluate the command language interface design. After this milestone, strand two will re-implement the standard visual interface in Java based on the XML handling routines, XML-level data model API, and display and interface library routines defined during the first year, complete the implementation of the command language interface, and build an overall user interface which links access to the different user functionalities. The task allocations given in figure 3 will be adjusted at this time, including allocated those relatively small that are currently unallocated. Strand three will continue through the prioritized list of changes to The Observer. We intend both end user and developer evaluation to occur for the software resulting from this phase of the work.

8 Acknowledgements

We gratefully acknowledge the support of the NITE project by the European Commission's HLT Programme.

9 References

Beattie, G. W. (1985). The threads of discourse and the web of interpersonal involvement. *Bulletin of the British Psychological Society, 38*, 169-175.

Broeder, D., Offenga, F., & Wittenburg, P. (2001). *EAGLES/ISLE Overview of Metadata Initiatives and Corpus Metadata in Language Engineering and Linguistics* (ISLE IMDI Deliverable D10.1).

Carletta, J. C., McKelvie, D., & Isard, A. (2002 (to appear)). A generic approach to software support for linguistic annotation using XML. In G. Sampson & D. McCarthy (Eds.), *Readings in Corpus Linguistics*. London and NY: Continuum International.

Dybkjær, L., Berman, S., Bernsen, N. O., Carletta, J. C., Heid, U., & Llisterri, J. (2001). *Requirements Specification for a Tool in Support of Annotation of Natural Interaction and Multimodal Data* (ISLE NIMM Deliverable D11.2).

Noldus, L. P. J. J., Trienes, R. J. H., Hendriksen, A. H. M., Jansen, H., & Jansen, R. G. (2000). The Observer Video-Pro: new software for the collection, management, and presentation of time-structured data from videotapes and digital media files. *Behavior Research Methods, Instruments & Computers, 32*, 197-206.

Orestrüm, B. (1983). *Turn-taking in English Conversation*. Lund, Sweden: Gleerup.

Vintar, S., & Kipp, M. (2001). Multi-Track Annotation of Terminology Using Anvil, *Proceedings of the Workshop on "Multi-layer Corpus-based Analysis" (Eurolan 2001)*.

Appendix: URLS for XML standards and software

dbXML Core 1.0b1, native XML database, http://www.dbxml.org/

Document Object Model (DOM) Level 2 Core Specification Version 1.0, W3C Recommendation 13 November 2000, http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113. Latest version: http://www.w3.org/TR/DOM-Level-2-Core.

Java APIs for XML Processing (JAXP) Release 1.1, http://java.sun.com/xml/xml_jaxp.html.

Kweelt XML Query Processor, http://db.cis.upenn.edu/Kweelt/. Currently, Kweelt Deux is under development.

Quilt: An XML Query Language, http://www.almaden.ibm.com/cs/people/chamberlin/quilt.html.

SAX 2.0: The Simple API for XML, http://www.megginson.com/SAX/index.html.

SAXON, The XSLT Processor (version 6.4.4), http://saxon.sourceforge.net/.

Xalan-Java (version 2.2.D10), http://xml.apache.org/xalan-j/.

Xerces2 Java Parser, http://xml.apache.org/xerces2-j/.

XML Linking Language (XLink) Version 1.0, W3C Recommendation 27 June 2001, http://www.w3.org/TR/2000/REC-xlink-20010627/. Latest version: http://www.w3.org/TR/xlink/.

Fujitsu XLink Processor (XLip), http://www.labs.fujitsu.com/free/xlip/en/. This is only available as part of a demo specification. It doesn't implement full XPointer syntax, just barenames (..#anchor) and “child sequences”.

Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006. Latest version: http://www.w3.org/TR/REC-xml.

XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116. Latest version: http://www.w3.org/TR/xpath.

XPath Requirements Version 2.0, W3C Working Draft 14 February 2001, http://www.w3.org/TR/2001/WD-xpath20req-20010214. Latest version: http://www.w3.org/TR/xpath20req.

XML Pointer Language (XPointer) Version 1.0, W3C Candidate Recommendation 11 September 2001, http://www.w3.org/TR/2001/CR-xptr-20010911/. Latest version: http://www.w3.org/TR/xptr.

XML Query Use Cases, W3C Working Draft 08 June 2001, http://www.w3.org/TR/2001/WD-xmlquery-use-cases-20010608. Latest version: http://www.w3.org/TR/xmlquery-use-cases.

XML Query Requirements, W3C Working Draft 15 February 2001, http://www.w3.org/TR/2001/WD-xmlquery-req-20010215. Latest version: http://www.w3.org/TR/xmlquery-req.

XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999, http://www.w3.org/TR/1999/REC-xslt-19991116. Latest version: http://www.w3.org/TR/xslt.

XSL Transformations (XSLT) Version 1.1, W3C Working Draft 24 August 2001, http://www.w3.org/TR/2001/WD-xslt11-20010824/. This document is no longer updated and will be replaced by the XSLT 2.0 Requirements, which is not yet available