Augmented **M**ulti-party **I**nteraction
http://www.amiproject.org

Augmented **M**ulti-party **I**nteraction with **D**istance **A**ccess
http://www.amidaproject.org

State-of-the-art overview

Annotation and query of multimodal databases

Updated version 23.01.2007

1 Introduction

There are many tools around for annotating language corpora, but they tend to be good for one specific thing and they all use different underlying data formats. This makes it hard to mark up data for a range of annotations - disfluency and dialogue acts and named entities and syntax, say - and then get at the annotations as one coherent, searchable database. It also makes it hard to represent the true structure of the complete set of annotations. These problems are particularly pressing for multimodal research because fewer people have thought about how to combine video annotations for things like gesture with linguistic annotation, but they also apply to audio-only corpora and even textual markup. The open-source NITE XML Toolkit is designed to overcome these problems.

At the heart of NXT there is a data model that expresses how all of the annotations for a corpus relate to each other. NXT does not impose any particular linguistic theory and any particular markup structure. Instead, users define their annotations in a "metadata" file that expresses their contents and how they relate to each other in terms of the graph structure for the corpus annotations overall. The relationships that can be defined in the data model draw annotations together into a set of intersecting trees, but also allow arbitrary links between annotations over the top of this structure, giving a representation that is highly expressive, easier to process than arbitrary graphs, and structured in a way that helps data users. NXT's other core component is a query language designed specifically for working with data conforming to this data model. Together, the data model and query language allow annotations to be treated as one coherent set containing both structural and timing information.

Using the data model and query language, NXT provides:

- a data storage format for data that conforms to the data model
- routines for validating data stored in the format against the data model
- library support for loading data stored in the format; working with it and modifying it; and saving any changes
- a query language implementation
- libraries that make it easier to write GUIs for working with the data by providing data display components that, for instance, synchronize against signals as they play and highlight query results
- annotation tools for some common tasks including video annotation and various kinds of markup over text or transcription (dialogue acts, named entities, coreference, and other things that require the same basic interfaces)
- command line tools for data analysis that, for instance, count matches to a specific query
- command line tools for extracting various kinds of trees and tab-delimited tables from the data for further processing or more detailed analysis

2 NXT's approach to Data Modelling

NXT's main strength is in the approach it takes to data modelling. NXT is motivated by the need to have many different kinds of annotations for the same basic language data, for linguistic levels ranging from phonology to pragmatics. There are two reasons why such cross-annotation is prevalent. First, corpora are expensive to collect even without annotating them; projects tend to reuse collected materials where they can. Second, with the advent of statistical methods in language engineering, corpus builders are interested in having the widest possible range of features to train upon. Understanding how the annotations relate is essential to developing better modelling techniques for our systems.

Although how annotations relate to time on signal is important in corpus annotation, it is not the only concern. Some entities that must be modelled are timeless (dictionaries of lexical entries or prosodic tones, universal entities that are targets of referring expressions). Others (sentences, chains of reference) are essentially structures built on top of other annotations (in these cases, the words that make up an orthographic transcription) and may or may not have an implicit timing, but if they do, derive their timings from the annotations on which they are based. Tree structures are common in describing a coherent sets of tags, but where several distinct types of annotation are present on the same material (syntax, discourse structure), the entire set may well not fit into a single tree. This is because different trees can draw on different leaves (gestural units, words) and because even where they share the same leaves, they can draw on them in different and overlapping ways (e.g.,disfluency structure and syntax in relation to words). As well as the data itself being structured, data types may also exhibit structure (for instance, in a typology of gesture that provides more refined distinctions about the meaning of a gesture that can be drawn upon as needed).

The best way to introduce the kind of data NXT can represent is by an example such as the one shown in figure 2.

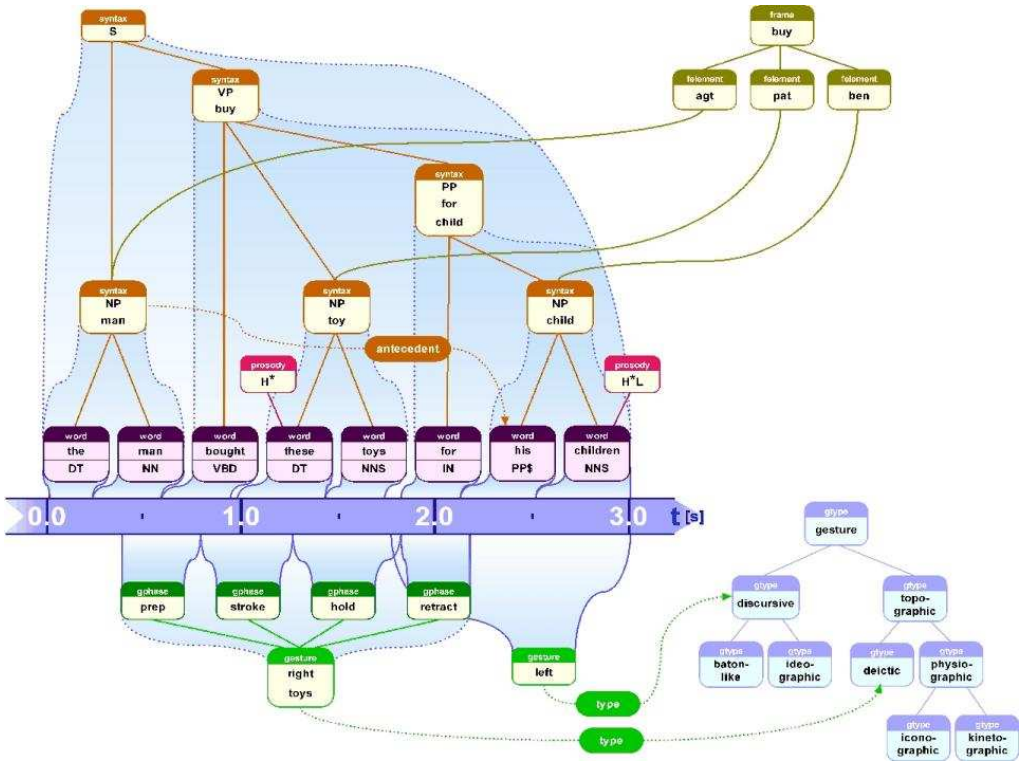

Figure 1: Example NXT data.

The picture, which is artificially constructed to keep it simple, contains a spoken sentence that has been coded with fairly standard linguistic information, shown above the representation of the timeline, and gestural information, shown below it. The lowest full layer of linguistic information is an orthographic transcription consisting of words marked with part-of-speech tags (in this set, the tag PP$ stands for "personal pronoun"). Some words have some limited prosodic information associated with them in the form of pitch accents, designated by their TOBI codes. Building upon the words is a syntactic structure – in this formalism, a tree – with a category giving the type of syntactic constituent (sentence, noun phrase, verb phrase, and so on) and the lemma, or root form, of the word that is that constituent's head. Prepositional phrases, or PPs, additionally specify the preposition type. The syntactic constituents are not directly aligned to signal, but they inherit timing information from the words below them. The very same syntactic constituents slot into a semantic structure that describes the meaning of the utterance in terms of a semantic frame (in this case, a buying event) and the elements that fill the roles in the frame (the agent, patient, and beneficiary). The last piece of linguistic information, a link between the syntactic constituent "the man" and the personal pronoun "his", shows that the former is the antecedent of the latter in a coreference relationship.

Meanwhile, the gesture coding shows two timed gestures and their relationship to a static gesture ontology. In the ontology, one type is below another if the former is a subtype of the latter. The first gesture, with the right hand, is a deictic, or pointing, gesture where the target of the pointing is some toys. This gesture is divided into the usual phases of preparation, stroke, hold, and retraction. The second gesture, made with the left hand, is discursive, but the coder has chosen not to qualify this type further. Gesture types could be represented on the gestures directly in the same way as parts of speech are represented for words. However, linking into an ontology has the advantage of making clear the hierarchical nature of the gesture tag set.

All of these kinds of information are used frequently within their individual research communities. No previous software allows them to be integrated in a way that expresses fully how they are related and makes the relationships easy to access. And yet this integration is exactly what is required in order to understand this communicative act fully. No one really believes that linguistic phenomena are independent; as the example demonstrates, deictic speech can only be decoded using the accompanying gesture. Meanwhile, many linguistic phenomena are correlated. Speaker pauses and hearer backchannel continuers tend to occur at major syntactic boundaries, an argument builds up using rhetorical relations that together span a text, postural shifts often signal a desire to take a speaking turn, and so on. The NITE XML Toolkit supports representing the full temporal and structural relationships among different annotations both as a way of keeping all of the annotations together and to allow these relationships to be explored, since understanding them should help our research.

Although the example shows a particular data structure that necessarily makes choices about for instance, how to represent coreferential relationships and what gestures to include in a taxonomy, NXT deliberately does not prescribe any particular arrangement. Instead, it is designed to be theory-neutral. NXT allows users to define their annotations and how they relate to each other, within constraints imposed by its internal data representation, the NITE Object Model.

3 Data Handling

In the NITE Object Model, annotations are described by types and attribute value pairs, and can relate to a synchronized set of signals via start and end times, to representations of the external environment, and to each other. Annotations can describe the behaviour of a single 'agent', if more than one participant is involved for the genre being coded, or they can describe the entire language event; the latter possibility is used for annotating written text as well as for interaction within a pair or

group of agents. The data model ties the annotations together into a multi-rooted tree structure that is characterized by both temporal and structural orderings. Additional relationships can be overlaid on this structure using pointers that carry no ordering implications. The same basic data objects that are used for annotations are also used to build sets of objects that represent referents from the real world, to structure type ontologies, and to provide access to other external resources stored in XML, resulting in a rich and flexible system.

Data is stored in a "stand-off" XML representation that uses the XML structure of individual files to mirror the most prominent trees in the data itself, forming 'codings' of related annotations, and pointers between files for other relationships. This has the useful properties of allowing corpus subsets to be assembled as needed; making it easy to import annotations without perturbing the rest of the data set; and keeping each individual file simple to use in external processing. For instance, the words for a single speaker can be stored in a single file that contains no other data, making it easy to pass to a part-of-speech tagger. NXT itself provides transparent corpus-wide access to the data, and so tool users need not understand how the data is stored. A 'metadata' file defines the structures of the individual files and the relationships among them, as well as detailing where to find the data and signals on the file system. Given this design, the data handling API includes routines for reading the metadata; for incremental data loading of an entire corpus, one 'observation' (corresponding to one text, dialogue, or meeting), or one XML file, including a 'lazy' option that only loads data when an application requires it; data access and editing methods; and methods for saving changed data. The loading methods give a choice in the level of data validation performed, and there is an off-line procedure for full validation against the corpus design expressed in the metadata.

4 Search

Search is conducted in NXT by means of a dedicated query language, NQL. Queries in NXT are reminiscent of first-order predicate calculus and express a set of variable bindings for data objects, optionally constrained by type, with any further conditions on the n-tuples to be returned expressed as boolean combinations of condition tests. The defined operators for the tests allow full access to the timing and structural properties of the data model as well as regular expression matching on the base text (or transcription) and on string-valued attributes. The language allows query results to be piped through another query for filtering, with the return value organized as a tree structure. The query language implementation evaluates queries expressed in NQL on a loaded data set, returning the data objects themselves for further processing, and can save returns in an XML format that can be co-loaded with the original data. A number of command line utilities implement the most commonly required search options, such as performing frequency counts and outputting tab-delimited data describing the search results. One such utility allows new annotations to be created from query matches. Analysis can also be performed by writing Java applications that use data loaded into the NITE Object Model and that either use the query language to search it or traverse the data using the object model API.

5 Support for writing GUIs

NXT contains a library of classes that can be used to build custom graphical user interfaces for annotation and analysis. This is a key part of its support for working with language corpora, since one of the major bars to progress in computational linguistics is the difficulty of creating data. The main components of the library are a media player; classes for display widgets that render, for example, text

and browsable tree structures; and utilities for operations such as loading and saving data. The media player is built on the Java Media Framework [7], which provides variable speed playback and moving backwards or forwards by seconds or frames, at least for some signal types. NXT can play multiple signals at once, subject to the memory limitations of the machine. All of the standard display classes give the option of highlighting the screen representations of data objects corresponding to current signal time in the player, allowing the annotation to be 'played' along with the media. The textual display includes implementation of an interface that allows it to highlight the screen representations of data objects that are selected in the search results display, which is useful for exploring the data and for authoring queries. Middleware is provided for the most common tool requirements, such as a text area that renders a standardized but configurable view of transcription. Together the library classes make it much easier to write effective tools for large-scale data annotation than could be done from scratch.

6 End user GUIs

There are some hand-annotation tasks so common that it makes sense to ship end user GUIs for performing them that are configurable for any given corpus or version of the annotation scheme. NXT currently comes with three such tools. The tools can either be used 'as is' or as model code for customized tools.

Continuous video labeller. The first is for creating timestamped labels against signal, with the labels chosen from an enumerated list. This can be used for a very wide range of low-level annotations, such as gaze direction, movement in the room, rough starts and ends of turns, and areas to be included in or excluded from some other analysis. The tool treats the labels as mutually exclusive and exhaustive states; as the user plays the signal, whenever a new label is chosen (either with the mouse or using keyboard shortcuts), that time is used both for the beginning of the new label and the end of the old one. Although there are several similar tools available, this tool will work on either audio or video signals, including playing a set of synchronized signals together, and works natively on NXT format data, which is of benefit for user groups that intend to use NXT for further annotation. It does not, however, currently include either waveform display or the palette-based coding displays popularized by Anvil [3], and is not currently intended for codings that require timings to the nearest video frame. That is, it has its uses, but it is not suitable for every kind of annotation.

Discourse entity and relationship coder. The second end user GUI is for coding discourse entities above an existing text or speech transcription. Coding is performed by sweeping out the words in the discourse entity and then mousing on the correct entity type from a static display of a type ontology, or by keyboard shortcut. It can be used for any coding that requires the user to categorize contiguous stretches of text (or of speech by one person) using labels chosen from a hierarchically organized set. In addition, it allows the user to indicate directional relationships between two coded entities, with the relationship categorized from another set of labels. The most common uses for this style of interface are in marking up named entities and coreferential relationships.

Discourse segmenter. The final GUI is for segmenting discourse into contiguous stretches of text (or of speech by one person) and categorizing the segments. The most common use for this style of interface is a dialogue act coder. Coding is performed either by sweeping the extent of the discourse segment or by marking its end; in the latter case, the segment is assumed to start at the end of the last segment or the end of the last segment by the same speaker, depending on how the tool is configured, and with the option of not allowing segments to draw words across some higher level boundary, such as previously marked speaker turns. A permanent dialogue box displays information about the currently selected act and allows a number of properties to be specified for it beyond simple type. The

coding mechanisms supported include ways of augmenting the main category choice using free text comments, tickboxes to cover boolean properties, and choice from enumerated property lists such as might be used for noting the dialogue act's addressee. Although this structure covers some styles of dialogue act coding, this tool is not suitable for schemes such as MRDA [6] where dual-coding from the same act type list is allowed. This tool additionally allows the user to indicate directional relationships between acts using the same mechanism as in the discourse entity coder, although for current dialogue act schemes this is a minority requirement.

Configuration of a tool is specified in XML; rendering choices such as what corpus element corresponds to a screen line are read from a tool configuration file, and annotation schemes specifying the category labels to use are described in stand-alone corpus resources. Under this arrangement, swapping in a new version of an annotation scheme is simple, although data sets do require careful management to ensure that each version of the data is loaded with the correct corpus resources. Figure 6 shows the discourse entity coder configured for named entities as used to create annotations for the AMI Meeting Corpus, [2, 1], as one illustration of NXT use.

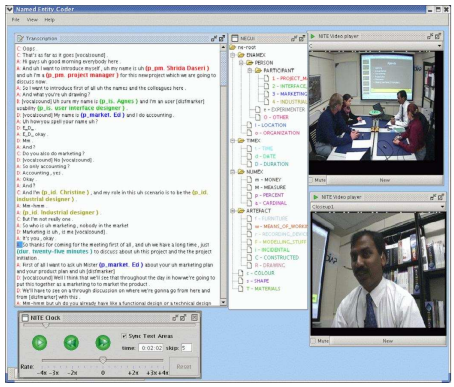

Figure 2: The discourse entity coder configured for named entities on a meeting corpus.

In addition to these configurable end user GUIs for specific tasks, NXT ships with a standardized data display and a standardized interface for running queries. The data display simply shows the media along with one window per corpus resource, ontology, and object set; one window for each coding, or set of related annotations, describing the entire interaction; and for codings that describe the behaviour of an agent, one window per agent. Within each display window, the primary tree structure, corresponding to the XML of an individual file, is shown, with traces for out-of-file child and pointer links. The query interface has one tab for query type-in and another for displaying results, which are shown in a browsable tree structure that gives full attribute-value details for data objects when they are selected. These interfaces will never be as usable as ones that display data and query results in a way that is tailored for a particular corpus and user community, but will at least work without further programming for any data expressed in NXT format. The software package also includes sample data and customized tools arising from previous users that can serve as model code for new customizations.

7 NXT's relationship to other tools

NXT has been used in conjunction with external tools for transcription, time-aligned annotation, and linguistic processing, by translating data among the various data formats required. NXT comes with

some import and export utilities, as well as more generic utilities for creating and recombining various trees from the data which makes this easier.

To our knowledge, NXT has not yet been used with other software packages that intend support for generic data handling and search such as the Annotation Graph Toolkit [5] and Atlas [4]. These frameworks currently have less complete implementations and less active development than NXT.

8 NXT's history and users

The NITE XML Toolkit is software that arose out of a European Commission-funded collaboration between the University of Edinburgh's The Language Technology Group [http://www.ltg.ed.ac.uk], the University of Stuttgart's Institut für Maschinelle Sprachverarbeitung (IMS) [http://www.ims.uni-stuttgart.de], and the Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI) [http://www.dfki.de]. Although the NITE project (NITE, IST-2000-26095) itself finished in 2003, the software is now being maintained and further developed via Sourceforge [http://sourceforge.net/projects/nite/]; the University of Twente [http://hmi.ewi.utwente.nl] has been a particularly active contributor. NXT is in use on a number of large distributed projects including JAST [http://www.jast-net.gr/] and TALK [http://www.talk-project.org/]. NXT is in use on a wide range of corpora, representing everything from Biblical text structure to the relationship between deictic expressions and gestures in multimodal refer- ring expressions. Its users range from individual PhD students up to large multi-site projects, many of whom contribute to development in some way. The AMI [http://www.amiproject.org] consortium is its biggest user and also the largest current contributor to its development. Other past and current funders are The Engineering and Physical Sciences Research Council (UK) [http://www.epsrc.ac.uk/] (EPSRC GR/T17663/01), The Economic and Social Research Council (UK) [http://www.esrc.ac.uk] (ESRC R022/25/0210), and Scottish Enterprise [http://www.scottish-enterprise.com/], via The Edinburgh-Stanford Link [http://www.edinburghstanfordlink.org/].

References

[1] AMI Consortium. AMI Meeting Corpus, 2006. http://corpus.amiproject.org/; accessed 13 Jan 2007.

[2] J. Carletta, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, W. Kraiij, M. Kronenthal, G. Lathoud, M. Lincoln, A. Lisowska, M. McCowan, W. Post, D. Reidsma, and P. Wellner. The AMI Meeting Corpus: A pre-announcement. In S. Renals and S. Bengio, editors, *MLMI'05: Proceedings of the Workshop on Machine Learning for Multimodal Interaction*, number 3869 in LNCS, pages 28–39. Springer-Verlag, 2006.

[3] M. Kipp. Anvil: the video research annotation tool, n.d. http://www.dfki.de/ kipp/anvil/; accessed 24 Jan 2007.

[4] C. Laprun, J. G. Fiscus, J. Garofolo, and S. Pajot. A practical introduction to ATLAS. In *Third International Conference on Language Resources and Evaluation*, Las Palmas, Canary Islands, Spain, 2002.

[5] Linguistic Data Consortium. AGTK: Annotation Graph Toolkit, n.d. http://agtk.sourceforge.net/; accessed 1 Mar 2004.

[6] E. Shriberg, R. Dhillon, S. Bhagat, J. Ang, and H. Carvey. The ICSI Meeting Recorder Dialog Act (MRDA) Corpus. In *HLT-NAACL SIGDIAL Workshop*, April-May 2004.

[7] Sun Microsystems. Java Media Framework API (JMF), n.d. http://java.sun.com/products/java-media/jmf/; accessed 1 Mar 2004.