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

(January 2006)

Abstract

The NITE XML Toolkit (NXT) is new open source software for working with multimodal, spoken, or text language corpora. It is specifically designed to support the tasks of human annotators and analysts of heavily cross-annotated data sets. Written in Java, it includes data handling and search; libraries upon which to base end user interfaces for hand-annotation and data analysis; configurable GUIs for some common hand-annotation tasks such as markup of named entities, coreference, and dialogue acts; and utilities for data transforms that make it easier to combine use of NXT with NLP applications and other corpus toolkits.

1 Introduction

Modern computational linguistics relies on large-scale data sets — sometimes text, sometimes speech, and increasingly often, multimodal — being marked up with many different phenomena at once. In this tool announcement, we describe a specific contribution towards the infrastructure required, the NITE XML Toolkit. NXT is new open source software for working with language corpora which is specifically designed for heavily cross-annotated data sets. NXT is aimed primarily at the needs of human analysts as they work with a corpus, for hand-annotation, automatic annotation where that relies on complex match patterns, the evaluation and hand-correction of automatic annotation, data exploration, descriptive analysis, and generating frequency counts as input to inferential statistics. It is currently in use on a number of research projects that, among other applications, have used it to create and work with named entities, dialogue acts, coreferences, topic segments, and extractive summaries on materials including written text, spoken dialogue, dialogue systems output, and meetings.

2 Data handling

In NXT, 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.

3 Data 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. One such utility allows new annotations to be created from query matches.

4 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 (http://java.sun.com/products/java-media/jmf/), 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.

5 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.

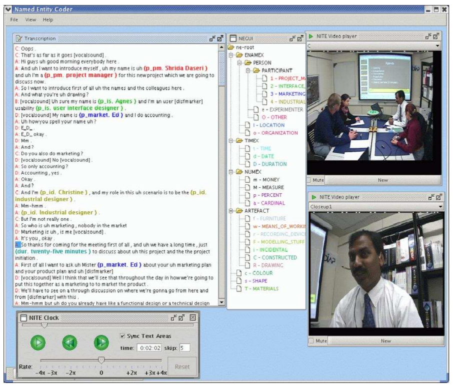
Continuous video labeller. The first is for creating timestamped labels against signal, with the labels chosen from an enumerated list. This can be used for a very wide range of low-level annotations, such as gaze direction, movement in the room, rough starts and ends of turns, and areas to be included 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 (http://www.dfki.de/ kipp/anvil/), 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 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 5 shows the discourse entity coder configured for named entities on a meeting corpus currently being annotated in the AMI project (http://www.amiproject.org/), as one illustration of NXT use.

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



C: Oops .
C: That's as far as it goes [vocalsound] .
A: Hi guys uh good morning everybody here .
A: And uh I want to introduce myself , uh my name is uh (p_pm. Shrida Daseri) and uh I'm a (p_pm. project manager) for this new project which we are going to discuss now .
A: So I want to introduce first of all uh the names and the colleagues here .
A: And what you're uh drawing ?
B: [vocalsound] Uh sure my name is (p_is. Agnes) and I'm an user [disfmarker] usability (p_is. user interface designer) .
D: [vocalsound] My name is (p_market. Ed) and I do accounting .
A: Uh howyou spell your name uh ?
D: E_D_ .
A: E_D_ okay .
D: Mm .
A: And ?
C: Do you also do marketing ?
D: [vocalsound] No [vocalsound] .
A: So only accounting ?
D: Accounting , yes .
A: Okay .
A: And ?
C: And I'm (p_id. Christine) , and my role in this uh scenario is to be the (p_id. industrial designer) .
A: Mm-hmm .
A: (p_id. Industrial designer) .
C: But I'm not really one .
A: So who is uh marketing , nobody in the market
D: Marketing is uh , is me [vocalsound] .
A: It's you , okay .
A: So thanks for coming for the meeting first of all , and uh we have a long time , just (dur. twenty-five minutes) to discuss about uh this project and the the project initiation .
A: First of all I want to ask uh Mister (p_market. Ed) about your uh marketing plan and your product plan and uh [disfmarker]
D: [vocalsound] Well I think that we'll see that throughout the day in how we're going to put this together as a marketing to to market the product .
D: We'll have to see on a through discussion on where we're gonna go from here and from [disfmarker] with this .
A: Mm-hmm but uh do you already have like a functional design or a technical design

Figure 1: The discourse entity coder configured for named entities on a meeting corpus.

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.

6 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, and comes with some utilities for creating and recombining various trees from the data, which makes this easier. To our knowledge, it has not yet been used with other software packages that intend support for generic data handling and search such as the Annotation Graph Toolkit (http://agtk.sourceforge.net/) and Atlas (http://www.nist.gov/speech/atlas/). There is no reason why this should be difficult, and it may be useful, as each will have its own strengths.

7 Distribution and Future Development

The NITE XML Toolkit is described more fully at http://www.ltg.ed.ac.uk/NITE/, including academic papers about the toolkit and its underlying data model, as well as a number of public reports about the experience of using it for a range of corpus applications. NXT is open source software and as such is intended as a community resource. The libraries and working samples are available from Sourceforge, as are active bug and feature request lists and bulletin boards. This means that anyone can participate in NXT's future development, by expressing their ideas for others to take up, by contributing code, or by investing resource earmarked against specific improvements. It is being taken up in a number of European and national projects that are just starting up, and there are currently several sites investing resource in it. Further development has already been pledged for better interoperability with other tools, memory management, improvements for timestamped coding including data displays that show time-alignment across different types of data, and integration with eyetracker output.

NXT has been developed primarily under European Commission funding (NITE, IST-2000-26095 and AMI, FP6-506811) with some UK research council support (ESRC R022/25/0210, EPSRC GR/T17663/01) and was based heavily upon the successes and failures of an earlier EC-funded prototype (MATE, LE4-8370). We are grateful to these funders and also to those who support Sourceforge and the Open Source Community.