The NITE XML Toolkit: data model and query language

Jean Carletta*, Stefan Evert**, Ulrich Heid**, and Jonathan Kilgour*

*University of Edinburgh, Human Communication Research Centre and Language Technology Group

**Institut für Maschinelle Sprachverarbeitung (IMS), Universität Stuttgart

RUNNING HEAD: NXT Data and Search

KEY WORDS: linguistic annotation, multi-modal language corpora, software tools

CORRESPONDING AUTHOR:

Jean Carletta

Human Communication Research Centre and Language Technology Group

2 Buccleuch Place

Edinburgh EH8 9LW

Scotland UK

J.Carletta@edinburgh.ac.uk

Tel: +44-131-650-4438

Fax: +44-131-650-4587

ABSTRACT:

The NITE XML Toolkit (NXT) is open source software for working with language corpora, with particular strengths for multimodal and heavily cross-annotated data sets. In NXT, annotations are described by types and attribute value pairs, and can relate to signal via start and end times, to representations of the external environment, and to each other via either an arbitrary graph structure or a multi-rooted tree structure characterized by both temporal and structural orderings. Simple queries in NXT express variable bindings for n-tuples of objects, optionally constrained by type, and give a set of conditions on the n-tuples combined with boolean operators. The defined operators for the condition tests allow full access to the timing and structural properties of the data model. A complex query facility passes variable bindings from one query to another for filtering, returning a tree structure. In addition to describing NXT's core data handling and search capabilities, we explain the stand-off XML data storage format that it employs and illustrate its use with examples from an early adopter of the technology.

1. Introduction

As language engineering becomes more sophisticated and moves into multi-modal applications, language corpora are becoming more complex. Corpora are currently being annotated for many different communicative phenomena: syntax, semantics, coreference, intentional structure, prosody, nods, gesture, deixis, and so on. Often the same basic material is annotated for many of these things. This naturally raises the analytical question of how the annotations relate to each other, either in theory or in the terms of descriptive statistical models. Thus, the community needs corpus infrastructure — data handling and search — suited to heavily cross-annotated data.

The NITE XML Toolkit (NXT) is open source software that is designed for the search and exploitation of multi-modal language data. It is designed to suit the needs of human analysts as they work with a corpus, for tasks such as 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.

We describe our data model, query language, storage format, and software implementation, using as an example data and queries from an early adopter of NXT, and place our work amongst similar toolkits being developed elsewhere.

2. Data handling in the NITE XML Toolkit

The NITE XML Toolkit provides library routines for loading and saving data in a rich data model that includes structural and temporal relationships among data objects, for

accessing the loaded data, and for changing it. Before we can describe this functionality in any more detail than this, we must describe the data model that it implements. A review of the approaches taken in other software is deferred to section 8 in order to make the comparison to NXT clearer.

2.1. The overall structure of a corpus

A 'corpus', or 'data set', consists of a set of 'observations', each of which contains the data for one interaction — one dialogue or small group discussion, for example. The interaction type collected in a corpus will involve a number of human or artificial 'agents'. Each observation is recorded by one or more 'signals', usually consisting of audio or video data for the duration of the interaction but potentially with information from other sources (blood pressure monitors, lie detectors, etc., as the research question demands). In the corpus, the signals are identified as recording a specific agent or as recording the interaction as a whole. Our technology assumes that all signals for an observation are synchronized, so that the same timestamp refers to the same time on all of them, as is usual with modern recording techniques.

2.2. Example motivating our data model

Now we turn to the annotation of an observation. The constructed example in figure 1 contains familiar linguistic structures from a range of annotations and both motivates and illustrates our overall data modelling approach. We have presented a general argument for this style of approach elsewhere (J. C. Carletta et al., 2004). In the figure, time is represented by the band running from left to right. Two kinds of annotation are time-aligned directly against signal, the orthographic transcription and the gestures. Timings are given either for a whole gesture or broken down into the

characteristic phases. More detailed gesture descriptions are structured in a tree where each level specifies subcategories of the level above, allowing the user to typify gestures at whatever level of description is convenient. Meanwhile, a syntax tree stands on top of the orthography, with a semantic frame built around some of the syntactic constituents. The syntactic constituents at least can reasonably be assigned timing information by inheritance from the words they dominate. Finally, a coreferential relationship is indicated between the noun phrase 'the man' and the possessive pronoun 'his'. NXT allows the representation of structures such as this one, but does not prescribe any particular representation for specific kinds of linguistic data. This example contains annotations for one agent but real data sets will contain annotations for every agent individually as well as, potentially, annotations that describe the interaction as a whole (for instance, the phases of a task-oriented dialogue).

PUT FIGURE 1 ABOUT HERE

2.3. Data objects

NXT's data handling employs data objects, each of which corresponds to one block from figure 1. Each data object has a type, which is a simple string. In addition, it may have any of four further kinds of properties. The first is a set of attribute-value pairs that further describe the data object, where the attribute is a string and the value can be a string or a number. The second is timing information, with start and end times relating it to signal. The third is a set of relationships to other annotations linking them as parent and child or via a pointer labelled with a named role, expressed

as a string. The fourth is textual content. Data objects may be related to other objects as their parents or may have textual content, but not both. The meaning of parenthood is that the children together make up the parent, whereas roles just convey some general connection. NXT requires paths constructed by following only child links not to contain cycles, but places no such constraint on pointers.

Data objects are primarily used in NXT to represent annotations, which are descriptions of some segment of the data, such as the words, syntactic constituents, gestures, and semantic frames that describe the text or language event under study. However, they are also used for 'objects' that represent something universal, like the things in the room at which the agents can point or the lexical entries in a dictionary. When data objects are used in this way, they cannot include timing information. Annotations can point to objects or other annotations using pointers with named roles, and objects can also point to objects or annotations. Data objects can themselves be organized into hierarchies using parent-child relationships or into more complex structures using pointers with named roles. One common use of objects is to create complex type ontologies that annotations can point to in order to augment their basic string types, as is done for gestures in figure 1. Figure 1 also shows a pointer used to show the relationship between two annotations, a pronoun and its antecedent, although this is only one of several possible representations for this information allowable in the data model.

The children of data objects are naturally subject to two kinds of ordering: temporal, based on the timing properties, and structural, based on ordering of the child list. Both types of order are important properties for data search. NXT maintains temporal

order, and to do so it requires annotations that are children of the same annotation not to overlap. In the data handling, intuitively, annotations may inherit timings from descendants that have been aligned directly to signal, but are not constrained to do so, allowing temporal and structural order to differ from each other. This is useful, for instance, when representing phenomena such as discontinuous syntactic constituents. The behaviour of the model is more completely described in (Carletta, Kilgour, O'Donnell, Evert, & Voormann, 2003) and formally defined in (Evert et al., 2002).

3. Data search

The real strength of the NXT data model is in the analyses it admits. Because different data sets can have widely varying structures, different users will wish to count and compare different kinds of things. This means that we need to support many forms of statistical analysis, not just simple frequency counts or timing measurements. The key to any analysis is data query. Thus our focus is on allowing the construction of queries that will return any of the kinds of things that users might wish to count and compare in data that fits our data model. For this we have defined a specialist query language, NQL, designed to exploit the relationships the model can express. We describe the query language here; query examples are given in section 7.3.

3.1. Simple queries

A simple query finds n-tuples of data objects (annotations and objects) that satisfy certain conditions. The query expression consists of two parts, separated by a colon (:). In the first part, variables representing the data objects are declared. These either match all data objects (e.g. '($x)' for a variable named '$x') or are constrained to draw

matches from a designated set of simple types (e.g. '($w word | sil)', matching data objects of the simple type 'word' or the simple type 'sil'). The second part of the query expression specifies a set of conditions that are required to hold for a match, which are combined using negation (logical not, '!'), conjunction (logical and, '&&'), and disjunction (logical or, '||'). Queries return a structure that for each match lists the variables and references the data objects they matched. For example, for a corpus containing 'word' data objects, the query

```
($w word):($w@pos = "N") && (TEXT($w) = "banana"))
```

will return a list of 1-tuples of words where the pos attribute has the value 'N' and the textual content of the word is 'banana'. The essential properties of a data object beyond its type are its identity, its attribute-value pairs, its textual content, its timing, and its relationships via the two types of structural links (child and pointer). Accordingly, NQL has operators that allow match conditions to be expressed for each of these kinds of things.

The attribute and textual content tests include the ability to match all data objects that have a particular attribute, regardless of value, or to match against the value of the attribute. Textual content is specified using a function, TEXT(), rather than, e.g., as $a@text, as if 'text' were an attribute name. This is to avoid potential naming conflicts with actual attributes in the data and because the meaning is properly functional. For data objects that have children instead of direct textual content, TEXT() returns the textual content from the data object's descendants, concatenated in the usual left-to-right transversal order. Attribute values or textual content can be

tested against given strings or numbers, or against the values found in other attributes or textual content, using equality, inequality, and the usual ordering tests (conventionally represented as <, <=, >=, and >). String values can also be tested against regular expressions. These allow matches based on templates that constrain parts of the string without specifying the value completely.

The temporal operators include the ability to test whether a data object has timing information, and to compare the start or end time with a given point in time. The query language also has operators to test for some common timing relationships between two data objects, such as overlap.

The structural operators test for dominance, precedence, and relationship via a named or unspecified role (including the reserved role used to designate complex types). One data object dominates another if it is possible to reach the second from the first by tracing zero or more child links. One data object precedes another if there is a structural ordering in which the first comes before the second, reckoning in the usual left-to-right tree ordering.

In addition, identity tests can be used to avoid matches where different variables point to the same data object. It is also possible in NQL to 'bind' variables within the query using existential ('exists') and universal ('forall') quantifiers in the variable declarations (which have the same meaning as in first-order logic). Such bound variables are not returned in the query result.

3.2. Complex queries

As well as simple queries, NXT supports the sequencing of queries into a 'complex' query using a double colon (::) operator. When two queries are sequenced in this way, the first query is evaluated in the usual way. Then its results are subjected to the further matching specified in the second query, which may use the variable bindings from the first query as well as the ones it introduces. The results for a complex query are returned not as a flat list but as a tree structure. In the tree, the children of the root are matches to the first query in the sequence. When a non-root node has children, these children give matches for the next query in the sequence that build on the parent match. Thus if there are n queries in the complex sequence, the result tree will be n+1 levels deep, if one counts the root node. Matches at any level for which there are no matches at the next are removed from the result tree completely, resulting in a tree of even depth. This produces a filtering effect which is of real advantage when exploring a data set, as we shall illustrate in section 7.3. For example, in a corpus of timed words from two speakers, A and B,

($wa word):($wa@agent = "A")::($wb word):($wb@agent="B") && ($wa # $wb)

will return a tree showing word overlaps; underneath each top level node, representing an overlapping word from speaker A, will be a set of nodes representing the words from speaker B that overlap that word of speaker A.

3.3. Discussion

Although NQL has the advantages that it is relatively intuitive and is flexible in being able to return n-tuples and tree structures, it is only one possible approach that is admittedly non-standard. We could have extended XPath (Clark & DeRose, 1999)

with additional axes to mirror the data model, but we ruled this out because the basic syntax would have made dealing with n-tuple and tree-structured returns cumbersome. In addition, current XPath implementations would have been of limited help because the extensions would require different underlying data structures. XQuery (Boag et al., n.d.) provides a more likely basis for implementation, but our less computational users would have difficulties using it to author queries.

4. Data storage format

NXT stores data in a set of XML files that relate to one another; this type of data representation is often referred to as 'stand-off' XML. XML files are essentially tree-structured, and in the data storage, the XML trees mirror the major tree structures in the data itself. In order to achieve this property in the serialization, even though the data objects alone form the basis for accessing, manipulating, and searching data in the API, NXT users must impose an organization on them in an overall data set design. The way that annotations are organized also makes it easy to ensure that there will be no cycles when following child links.

4.1. Organization of data objects

Annotations are organized into 'layers' consisting of all of the annotations for a particular agent (or for the interaction as a whole) that are either of the same type or are drawn from a set of related types and which together span the observation in some way, drawing together, say, all the words Fred spoke, or all the arguments the group had. Layers can be 'time-aligned', in which case the annotations in them relate directly to signal; 'structural', in which case they have a structural ordering; or 'featural', in which case they do not. Layers draw children (if any) from a single,

designated 'lower' layer. Structural layers may be specified to inherit timings from time-aligned children, in which case they may have structural and temporal orderings that differ. Layers can be defined to be recursive, in which case there will be recursive descent via children down one or more copies of layers with the same structure before before the lower layer is reached. The layer structure is what prohibits cycles.

Layers themselves are organized into 'codings', or sequences of layers where each layer draws children from the next one, again for a particular agent or for the interaction as a whole. Each layer must be placed in exactly one coding, but the last layer in a coding can draw children from a layer in a different coding. This arrangement enables the multiple parenthood that is a key property of the data model. By definition, the set of annotations that fall into a single coding form a tree structure. These are the trees that are mirrored in the XML, where each element represents one data object. Each interaction coding is stored in its own file, and agent codings are stored in one file per agent.

Like the set of annotations, collections of objects, called 'corpus resources', are defined using layers. NXT has in the past differentiated flat, one layer corpus resources from tree-structured ones by calling the former 'object sets' and the latter 'ontologies', but the distinction is uninmportant in practice. Corpus resources apply to the entire corpus, not single observations, and are stored in one file each.

4.2. The data format

In the XML, the element name gives the data object's type. Attribute-value pairs and textual content are represented in the obvious way, with any relationship to signal given using attributes that specify start and end time offsets. Where the metadata specifies that the children are drawn from a layer in the same coding, they will be represented by XML children. Children drawn from a layer in a different coding are represented using a special-purpose XML element. It specifies how to find the children using a stand-off link based on filename and ID, which every element is required to have. The arrangement for codes that point to other codes using named roles is similar, but uses a different element that additionally specifies the role. By default, these elements are called nite:child and nite:pointer, but these names are configurable.

This representation may sound awkward, but it is sensible. Assuming that the data does not fit neatly into one tree but that one wishes to keep the data in one file, there are two main possibilities. First, one can place the information in the single largest tree structure in the data, adding the crossing structures via in-file references, as the Penn Treebank does (Marcus et al., 1994). However, the more out-of-tree links there are, the more cumbersome and difficult to maintain such a structure becomes, and NXT is aimed at heavily cross-annotated data sets. Second, one can fall back on a list of nodes and links that make up the data's graph structure, as in TIGER's XML (Tiger Project, n.d.) or Atlas Interchange Format (*ATLAS Project*, 2000). The main disadvantage here is in making use of what is essentially an unstructured data representation outside TIGER itself. Tree structure is common in linguistic data, and mirroring at least some of it in the XML tree structure itself makes it easier to process using standard technologies as well as making it more human-readable.

One further option would be representing the data essentially as it is in NXT, but in one file. In this arrangement, codings, and corpus resources would be represented as at present but with links between codings expressed using idrefs. The first few levels of the tree down from the root would have a fixed structure organizing the data. Although this representation shares much in common with the one that NXT uses, splitting up the data has several advantages over this arrangement. First, separating the tags into their various types makes it easier to add data using external processes (part-of-speech taggers, named entity recognizers, and the like) because each individual file is itself simple. Second, different people can change different data files at the same time without conflict, as long as none of them edit the files they point to and all are able to lock complete paths of files pointing to the data they are revising. Third, a data set can readily be loaded whole or in part, speeding up some processing. Because NXT treats data relationships derived from child links and normal XML child structures in exactly the same way, how the data is divided into codings only matters whilst the data is being loaded or serialized. Meanwhile, cases where the data does fit neatly into a single tree can always be defined in NXT as a single coding, avoiding unnecessary complexity.

4.3. The metadata format

To use the NITE XML Toolkit's data handling, the data set designer must first write a 'metadata' file that formally describes the data set. The most substantive part of the metadata describes the mathematical structure of the annotations that the data set contains. This description is in the terms of the data model and organization; that is, in terms of a number of observations, signals, agents, corpus resources, codings,

layers, and data objects. Corpus resources and the element names they include must be declared in the metadata so that NXT knows to load them, but their structure need not be defined there as long as the resource does not itself contain out-of-file child or pointer links. This is to allow the loading of arbitrary XML files as corpus resources without undue effort.

Besides specifying the structure of the annotations, the metadata file also contains a number of other kinds of information about the data set. The most important of these are as follows. First, it holds information about where on the computer to find the files containing the signals and the annotations as well as any documents describing the data set. Rather than listing XML files individually, the correct filenames are constructed from paths, file extensions, observation, agent, coding and signal names specified in the metadata. Second, it specifies the names that the data set uses for things like the attributes of a element that hold its start and end times and id, and the names of the child and pointer elements, giving a degree of flexibility to the storage format itself. It does not allow one to record information corresponding to any of the emerging metadata standards (Wittenburg, Broeder, & Sloman, 2000) for describing in general terms what sort of material the data set contains, although it is possible to note where such metadata resides. This is because standard metadata does not need to be accessible to NXT itself, but to a higher level application for browsing the set of known data sets.

4.4. Relationship to standards

Where possible, software tools should adhere to any relevant standards for their data storage formats. By devising our own format, we have left ourselves open to criticism. Our reasoning is as follows.

For NXT's metadata, the part of the file that declares the structure of a corpus has the same functionality as a set of document type definitions (either DTDs or some variety of schemas) corresponding to the individual coding files. However, NXT needs to be able to inspect and enforce not just the document type definitions for whatever codings might be loaded, but also to do the same for cross-document relationships between codings. Document type definitions cannot express this information, which means that we would need a special purpose mechanism for this. Meanwhile, no existing libraries parse DTDs in a way that exposes them to inspection. XML Schema is at least amenable to XML parsing, but admits so many ways of specifying the same constraints that inspecting an arbitrary schema to determine, for instance, the permitted values for a given attribute is relatively difficult. Because our content models are relatively simple, with a set of permitted data object types in each layer, it is easy to devise a simple format that yields the information readily. For this reason, we define our own, using it for inspection and validation within NXT, but provide a utility for generating a schema from it for full off-line validation (see section 6).

Meanwhile, there is currently no mature standard suitable for storing data annotations that provides what we need. ISO (ISO/TC37/SC4, n.d.) intends to define standards for the representation of data in language processing, which could be a good match for a subset of NXT's users, but is still in the early stages of deliberation. Other related standards for the representation of linguistic data are the Text Encoding

Initiative (TEI Consortium, 1999) and the current working draft of EMMA (W3C, 2005). We did not use TEI recommendations for our storage format because the TEI, which has its greatest strengths for text, is agnostic about concerns of importance for our data modelling such as how to keep timing information coherent across a data set. In addition, stand-off annotation is critical for projects in which several sites annotate the same material at the same time for different properties, but the TEI currently employs a one-file storage format. On the other hand, EMMA is aimed at those developing applications that handle multimodal user inputs, and is a standard for representing the effects of machine processing on one multimodal input at a time. Although these can be seen as machine annotations of the data much like the hand annotations prevalent among NXT users, EMMA has no treatment of dialogue context. This makes EMMA unpalatable as an alternative to NXT's representation. However, good translation between the two formats is likely to become important to some NXT users in future.

5. The implementation

The NITE XML Toolkit is implemented in Java and makes use of Apache's Xerces (The Apache XML Project, n.d.) for its underlying XML handling. We describe the data handling and search libraries here, but also briefly describe the rest of the toolkit as it is important in many uses of this software and contributes to the worked example in section 7.

5.1. The Data Model Implementation

The implementation provides a library of Java routines for loading data, saving it, accessing what has been loaded, and making changes. The implementation can handle

child and pointer stand-off links in either a format conformant with the W3C XLink (DeRose, Maler, & Orchard, 2001) and XPointer (DeRose, Maler, & Daniel Jr., 2001) standards or the pre-standard syntax employed by LT-XML (The Language Technology Group, n.d.). The link style is specified in the metadata file for a corpus.

Once the metadata for a data set has been loaded, the data itself can be loaded either entirely, for a single observation, or for a particular set of codings. How to stream this sort of data model is a research issue in its own right. The current implementation requires complete coding files to be loaded at a time and uses memory equivalent to five to seven times the data's storage size on disk. It is possible to load individual codings from a data set one at a time as they are needed, improving performance and reducing memory use that way. However, loading more than one observation at a time is currently problematic for more complex corpora. The API includes methods for loading multiple versions of the same coding in order to support reliability analyses and comparisons between hand and machine coding of the same information. One can save either to the locations on disk from which the annotations were loaded or to a new one; there is also a choice about whether to save everything that was loaded or just the codings that have changed.

Typical data access routines allow one to iterate over all of the loaded data objects or to find an annotation's start and end times, parents, children, and roles. Typical routines for changing the data create new data objects (which are assigned to a particular layer depending on their type and the agent whose behaviour they describe), delete data objects, modify a data object's attributes, and link two data objects using the parent/child structures and pointers. The metadata implementation enables

complete access to the information in the metadata, but at present manipulation is limited to those edits that are most likely to be needed programmatically: changing the paths to the files making up a data set, adding observations, and recording data management information. Other types of edits can easily be made in an XML editor.

NXT's data representation requirements are expressed in rather stringent terms. Some requirements are necessary in order to make the data model work, such as the one that any two elements in the same time-aligned layer must not overlap temporally. Others facilitate data loading and serialization (for instance, the same data object type cannot appear in two different codings). These kinds of requirements cannot be violated without breaking the system. However, another class of requirements are less serious, and in fact, many of them can be violated with no (or limited) loss of functionality. For instance, NXT requires every XML element to have an ID in case the application should choose to reference it in a stand-off link, but in practice the loader adds IDs to any element that does not have one, and these new IDs will be saved with the data set. The NXT metadata format's use of codings and layers imposes a stricter structuring of the data than treating the data as a graph of data objects. In general, as long as a data set keeps the data graph coherent and the metadata gives a clear mapping from data object types to codings so that the data can be serialized, it will work in the implementation.

5.2. The Query Language Implementation

The main function of our query library allows a query to be evaluated on a loaded data set, and can either return all matches or cut off when some given maximum number is reached. The simpler queries execute quickly enough for interactive

operation, but some, such as those that employ 'forall', are currently best executed as batch jobs; past users have written queries that take up to 15 seconds to run per observation or three hours over an entire corpus. Query results are returned as list structures amenable to further processing. The implementation also supports saving results in a simple XML format that points into the data. By adding a corresponding coding definition to the metadata, query results can then be loaded as part of the data set itself. Another option allows results to be saved in an Excel spreadsheet. This option is sometimes preferred by less computational users. The current implementation performs a left-to-right depth-first traversal of the result tree, creating one spreadsheet row per match list. This is human-readable but makes the output for complex queries unsuitable for onward processing by obscuring the tree structure.

5.3. Utilities and graphical user interfaces

A number of programs are distributed with NXT that serve both as example applications and as generic corpus utilities in their own right. One performs frequency counts for query matches, reporting the results either for a complete corpus or for each observation individually. Another extracts tab-delimited output from a set of query results based on user-defined templates. An indexer adds new data objects to a corpus that point to matches to the first named variable in a given query. Other utilities based on the concept of 'knit' from LT-XML (The Language Technology Group, n.d.) will produce from a coding the larger XML tree structure that can be obtained by inserting traces for the targets of pointer links or by recursively replacing child links with the material to which they refer. The utilities provide useful mechanisms for transforming the data for consumption by external processes such as part-of-speech taggers. More difficult transforms can be accommodated using a

variety of XML processing techniques or by loading data into NXT's data model, traversing it, and writing non-NXT output.

A major part of NXT, although it is not featured here, is a library based on Swing and Java Media Framework for use in building tailored text-based data displays and coding interfaces. NXT illustrates use of this library through a number of sample applications that work on its data examples. In addition, NXT comes with two graphical user interfaces that will work on any NXT-format corpus without any programming effort. The first of these, NXT Search, allows search over an entire corpus; within it, the user can load a corpus, type in a query, and display the results. The display simply shows the result tree, allowing the user to hide or show the children and detailed variable bindings for each node in the tree, and tells the user how many matches there are at each level. There are extensive help screens and from the interface it is possible to save the query results. NXT Search is available with the NXT library distribution but also as an easy-to-install stand-alone application and in a web demo (*NXT Search*, n.d.). The second utility loads a single observation and yields a data display with separate windows for each coding and corpus resource, with rudimentary information about links to other files. This utility has a search facility that uses the NXT Search GUI, but with the addition that selecting part of the query result tree will highlight the corresponding elements on the data display.

6. Data validation

Although often overlooked, data validation is such an important process for preserving the integrity of the data set that it deserves description. When data validation is insufficiently stringent, incompatibilities with the underlying model can

arise, leading to unexpected behaviour. On the other hand, if validation is too rigourous during time-critical operations this can make applications unusable. In the NXT implementation, validation whilst editing is stricter than validation whilst loading. This is for two reasons. First, the amount of data to be handled at load time can be very large compared to that during individual edits. Second, it should be possible in some circumstances to load and then fix a broken data set, but impossible to move from valid to invalid data using the API.

During data edits made using the API, the NXT data model is strictly enforced. When an edit attempts to add a pointer or parent/child relationship to the data, the implementation checks that the metadata allows the elements involved to be in the specified relationship before carrying out the edit. This includes checking the names of pointer roles. Attribute edits fail unless the specified value is of the correct data type. In addition, start and end time edits are checked to make sure they are compatible with the temporal ordering of the data model, so that, for instance, an element's start time cannot be set to a time before the end time of its preceding sibling.

By default, the data loading implementation strictly checks the types of the attributes that are important for structural integrity — IDs and start and end times — and fails to load elements and attributes if they are not described in the metadata, but otherwise merely warns about non-compliance with the data model. However, structural validation can be switched on for data loading via the API for applications that can afford the overhead and prefer stricter checking.

Many users will wish to run standard XML processes over individual files from an NXT data set. For this reason, it is useful to validate data not just during processing in NXT, but also off-line. NXT's specialist metadata format is easy to author and it includes the equivalent of the set of document type definitions for the data set as a whole, but it cannot be used for formal off-line validation of the XML in which the data is stored. We facilitate this validation by providing a script that generates an XML schema from the metadata file that can be used to validate any XML file from the data set individually or any larger XML tree derived from the data by knitting (cf. section 5.3).

7. Example corpus treatment

When arguing for the utility of an earlier data model that is comparable to our own, the annotation graph, (Bird & Liberman, 2001) showed that the data model could be used to represent a wide range of existing linguistic corpora. Although this was a useful exercise, we do not repeat it here. Our data model admits arbitrary graph structures with a straightforward percolation of timing information up privileged parts of the structure. A more relevant question than whether the model can represent some data is how well the data model fits its intended uses. In this section, we describe the representation of a data set in NXT and use of the query language to analyze that data in order to expose the properties that result from our design. This data set has been chosen because it exercises most of the features of the data model. A complementary example, based on a project that is adding discourse annotation to the Switchboard corpus (Carletta, Dingare, Nissim, & Nikitina, 2004), has previously been reported; it does not relate annotations to the speech signal, but it does use pointers with named roles and more of the utilities than this one, including automatic annotation. Another

published example (Heid et al., 2004), a translation of the LeaP corpus (Milde & Gut, 2002), annotates 12 hours of native and foreign language learners' speech in English and German with eight different levels of description, including speech vs. non-speech, words (with part-of-speech and lemma information), syllables, segments, intonation, and pitch movements, and demonstrates the utility of treating annotations as structured rather than simply as time-aligned labels.

7.1. The MONITOR data set

The HCRC Map Task (Anderson et al., 1991) is a well-established dialogue elicitation task in which two subjects have similar maps that differ somewhat in what is marked on them and where one subject (the 'route giver') has a route marked on the map whilst the other (the 'route follower') does not. The task is for the subjects to discuss the maps in enough detail for the route follower to draw the route on his map. MONITOR uses variants of this task to determine the effects on the route giver's contributions of different levels of route follower feedback. In this particular example, which tests the minimal feedback case, the route giver is told that the route follower is in another room and can hear him but not be heard. The subject is eye-tracked (the trace shows up on the video as a white circle), and is told that the red square on the video indicates where the route follower is looking. However, the red square is actually manipulated programmatically. There is no route follower.

The data is orthographically transcribed and segmented into dialogue moves (which carry out one speaker intention) and transactions (which group moves into larger, plan-oriented chunks) as specified in (Carletta et al., 1997). Although words in this data set are not individually timestamped, moves are. References to map landmarks

have been identified in the transcription and timestamped. Areas of disfluency are also identified based on (Lickley, 1998), including a type classification (e.g., repetition, substitution) and separation of the reparandum and repair, but this information is related directly to signal using timestamps, not overlaid on top of the transcribed words.

In addition to this coding of the speech signal, there are several different time-aligned gaze codings that derive from the eye-tracker output. **Feedback** encodes information about the placement of the red feedback square using categories that one might expect to affect subject behaviour: the square can be at the correct landmark (i.e., the one to which the subject refers), at the wrong landmark, in motion, or not present. **Feedbackgaze** coding records whether the subject is looking at or away from the red feedback square. It is useful for determining whether or not he is attending the feedback well enough for it to affect his behaviour. **Routegaze** coding gives another interpretation of subject gaze that relates it to task planning. It categorizes where the subject is looking relative to the route, such as at the landmark associated with the part of the route currently being described, at one associated with a previous or future part of the route, or at some other landmark.

Figure 2 shows a screenshot of the NXT-based data display and search GUI in use by the MONITOR project, and may be helpful for understanding the structure of the worked example. For the sake of simplicity, it omits transactions, disfluencies, and referring expressions in the rendering, although the full data is accessible when querying. Both the display and the data for one dialogue are distributed as a sample with NXT.

PUT FIGURE 2 ABOUT HERE

7.2. Representing MONITOR data in NXT

Although MONITOR is technically a dialogue task, since there is only speech for the subject and since different codings are used for the subject and the (fake) dialogue partner, it is easiest to declare codings as representing the interaction as a whole. The NXT representation of MONITOR data uses one interaction coding for each of the three types of gaze coding. Each gaze coding contains one time-aligned layer that is terminal (i.e., where the annotations have no children). Because the gaze codings represent mutually exclusive and exhaustive states, together they span the length of the video signal, and the end time of one tag is the start of the next. Disfluency coding is similarly grounded in signal, but the layer is declared to be recursive, allowing for arbitrary levels of nesting in order to cover compound cases. Orthographic transcription is a coding of a single terminal layer, but in this case the layer is structural, not time-aligned. A referring expression coding is again a single time-aligned layer, but one that draws children from the transcription layer. Unlike for the gaze codings, referring expressions occur sporadically rather than spanning the entire signal. Finally, dialogue structure coding consists of two layers, transactions and their constituent moves, which again point to the transcription. The move layer is declared as time-aligned but the transaction layer as structural, allowing the move timings to percolate up to the transactions for use during analysis.

We show an extract from the most complex file, the dialogue structure coding, in order to give the flavour of the data format.

```
<nite:stream id='18t-ft.ds'>
  <transaction id='trans1' type='normal'>
    <move id='move1' type='instruct_move'  start='5.38' end='8.939'>
        <nite:child href='18t-ft.trans.xml#id(w1)'/>
        ...
        <nite:child href='18t-ft.trans.xml#id(w8)'/>
    </move>
    <move .../>
  </transaction>
  ...
</nite:stream>
```

In the extract, the <nite:stream> element just gives the XML document root and does not correspond to anything in the NXT data model. Children of this root node belong to the transaction layer, and their children, to the move layer; in turn, their children are specified using out-of-file nite:child links, which are specified using LT-XML link syntax in this corpus. In this syntax, '18t-ft.trans.xml#id(w1..w8)' is allowable as a convenient shorthand for the elements in 18t-ft.trans.xml between w1 and w8, which are required to be sisters. It is conventional to number IDs systematically, but the shorthand works regardless.

The corresponding structural definition in the metadata for transaction and move coding simply defines two layers within the same XML file, one for transactions and

one below that containing moves and the silences between moves. This portion of the metadata looks as follows.

```
<coding-file name="dial">
    <structural-layer name="trans-layer" points-to="move-layer">
        <code name="transaction">
            <attribute name="type" value-type="string" />
        </code>
    </structural-layer>
    <time-aligned-layer name="move-layer" points-to="ref-layer">
        <code name="move">
            <attribute name="type" value-type="string" />
        </code>
        <code name="ims" /> <!-- inter-move silences-->
    </time-aligned-layer>
  ... more layers...
</coding-file>
```

In the metadata extract, the type attributes would most properly be defined as enumerated, with the list of possible values given in the metadata, but here are defined as strings to save space. The metadata declares that moves and silences between moves are timed, with the times stored in attributes that are common over all codes in the corpus and therefore are declared elsewhere in the metadata. It also declares that transactions inherit their times from the moves and silences they contain.

7.3. Search using NXT

The easiest way to give a sense of what it is like to use NXT to search a data set is to give some simple query examples.

```
($m move):($m@type = "instruct_move")
```

matches all moves that are labelled as instructions.

```
($w word)(exists $r reference):($r ^ $w)
```

matches all words that are contained within referring expressions; the hat (^) operator specifies the dominance relationship.

```
($w word)(forall $r reference): (TEXT($w) = "the") && !($r ^ $w)::
    ($m move)(forall $d disf): ($m ^ $w) && !($d # $m)
```

matches all occurrences of the word 'the' that are not dominated by a referring expression and are dominated by a move (as all words are in this data set) that does not overlap temporally (#) with any disfluency (which might provide a reasonable excuse for the lack of reference markup). The words will occur at the first level of the result tree, with their dominating moves underneath. Although there may well be good reasons for matches to occur, exploring a data set using this sort of query can give the user a much better understanding of its character and quality.

The complex query facility in NQL has proved particularly useful for breaking down frequency counts for some phenomenon using a subquery to determine the bins.

Essentially, the first query expresses the conditions for the overall phenomenon, with the bins expressed in a subsequent query template which loops through the various possibilities. For instance, consider the case of wishing to count times that the subject looks at the feedback square, sorting the cases by what type of dialogue move is being uttered when the subject starts looking:

```
($g feedbackgaze):($g@label= "feedback")::
   ($m move):(START($g) > START($m)) && (START($g) < END($m)) &&
         ($m@type="LABEL")
```

Instantiating LABEL with each possible move type in turn yields the desired count. Queries that serve as parity checks are reassuring and relatively easy to devise. In general, using the :: operator has the advantage of allowing queries to be constructed piece by piece, testing each addition in turn by exploring the search results via the highlighting on the data display. Where one might use 'exists', there is often a complex query that yields similar results but uses an ordinary variable instead, and for this reason is easier to debug. Complex queries are usually easier to write and are currently faster to run than simple queries with the same terms that throw more variables and conditions together at once, and the hierarchical result lists that complex queries return are also easier to map to natural ways of counting events in the data.

Although transforming the original MONITOR data set into the NXT data format required someone who could write Perl scripts and XSLT stylesheets, staff without computational backgrounds have been able to use NXT for analysis. They have designed some surprisingly complicated queries by adding one set of conditions at a

time and checking their work using search highlighting on a graphical user interface. For instance, one of their 'best' queries counts cases where the subject is in the middle of saying something as part of a first attempt to cover some part of the route, and notices that the feedback square indicating the partner's gaze moves to the wrong place, dividing these cases by the purpose of the next dialogue move the subject makes (where the expectation is a preponderance of explaining moves). This previously would have been impossible without special purpose programming.

8. Other data handling mechanisms

Although there are many specialist applications that represent and handle linguistic data, support for generic data handling and search is rarer. The most comparable libraries to NXT are the Edinburgh Speech Tools Library (ESTL), EMU, the Annotation Graph Toolkit (AGTK), and Atlas. These libraries all define a data representation with an implemented API for data handling and at least have a story to tell about search. Except for EMU, they all store linguistic annotations in a single file and so fail to support distributed data creation or selective data loading. Support for data query in all of the libraries is currently patchy, although several of the efforts plan future development in this area. Although technically all of the libraries can represent arbitrary graph structures, all of them apart from Atlas and NXT focus on speed for high-volume speech applications, for instance, by processing annotations one 'utterance' at a time or by omitting data access based on structural properties from their APIs. This makes them less suitable than NXT for creating and working with higher level linguistic annotations, where larger discourse structures are required and the structural properties of annotations are of primary importance.

8.1. The Edinburgh Speech Tools Library

ESTL (*The Edinburgh Speech Tools Library*, n.d.) includes an C++ API for the 'heterogeneous relation graph', or HRG (Taylor, Black, & Caley, 2001), created with text-to-speech systems in mind but proposed for general use. The data structures employed represent complete sequential lists, trees, or multi-linear structures for a single utterance, usually defined as one stretch of speech to be realized. Functional values can be used to derive the timings of an annotation from related data. Technically, this is as flexible as NXT's object-based approach, as long as one is willing to define an utterance as containing all of the annotation for a complete signal from all speakers and to decompose non-tree-structured graphs into as many multi-linear structures as might be required. However, representing the data in this way is non-intuitive and loses the main advantage of the HRG formalism, which is efficiency in traversing the graph structures it is designed to represent. Acoustic information such as F0 contours, essential for text-to-speech, can be stored using either the list data structure or a different, more efficient frame-based representation. This kind of information is not common in the sorts of databases that represent higher-level linguistic properties but could be useful. The library does not include a specific query facility; since Festival, a speech synthesis system, uses a Scheme interpreter to call the API functions, (Taylor et al., 2001 p. 173) suggests making use of the Scheme bindings to write queries in Scheme directly. It is unclear whether this approach has been used in practical work or whether HRGs have been employed outside speech synthesis. HRGs are stored in a dedicated format but can be exported and read to and from other formats including one in XML.

8.2. The EMU Speech Database System

EMU (Cassidy & Harrington, 2001; *The EMU Speech Database System*, 2001), also aimed primarily at speech researchers but proposed for general use, places annotations in levels according to linguistic type. Annotations within a level can overlap and need not span the signal, but levels are partially ordered temporally. Annotations, whether on the same level or on different levels, can be related by dominance and association, which have the same basic meanings as parent/child relationships and pointers in NXT, respectively, except that associations do not have types and there is no notion of untimed data such as is required for representing corpus resources. EMU can split the storage of different levels for different utterances into different files. It also uses 'database templates', similar to NXT's metadata file, to specify where to find the data and the permitted relationships among levels for a specific database, an important first step towards validation. However, the emphasis in EMU is clearly on temporal order and dominance; association is not supported in the labelling tool that comes with EMU for creating annotations, and most of the external data formats it supports treat annotations as simple time-aligned labels. The query language supports neither relationships between utterances nor association, a reasonable decision given the original target user group. EMU is written in C++ and provides a set of extensions for the Tcl scripting language, a facility that has been used to write several graphical tools, but it does not contain libraries that specifically support the development of tailored GUIs. (Cassidy & Harrington, 2001 p. 75-6) suggests that the kinds of user interfaces required for creating high-level linguistic annotations are so different from the EMU labeller that it would be better to create them using other tools, although they could then be loaded into the EMU database for query.

8.3. The Annotation Graph Toolkit

AGTK (*AGTK: Annotation Graph Toolkit*, n.d.), employs a data model, the annotation graph (Bird & Liberman, 2001), which is a directed acyclic graph where edges are labeled with feature-value pairs and nodes can be labeled with time offsets. Structural relationships can be expressed by convention in the edge labelling, but, with the exception of an API for handling tree structures, they are not exposed directly in the API as they are in NXT; instead, the focus is on the efficient handling of temporal information. AGTK is written in C++ and comes with a Java port. The toolkit directly supports a range of data formats but the primary one is XML that describes the graph structure of the data rather than using the XML structure to mirror its major trees as NXT does. A query language is planned for AGTK but is not yet available. Although AGTK does not provide direct support for writing graphical user interfaces, it does include wrappers for Tcl/Tk and Python, two scripting languages in which writing such interfaces is easier than in C++ itself. AGTK-based tools have been used to support an initiative for rapid creation of linguistic resources (Strassel, Maxwell, & Cieri, 2003).

8.4. ATLAS

Atlas (*ATLAS Project*, 2000; Laprun, Fiscus, Garofolo, & Pajot, 2002) is intended to generalize the annotation graph and differs in two main ways. First, it allows richer relationships between annotation and signal. In annotation graphs, the only relationship between annotation and signal that is supported in the data handling is the timespan on the signal to which the annotation refers, given as a start and end time. NXT is similar to AGTK in this regard. Atlas, however, defines more generic signal regions which can refer to other properties besides the timing. For example, on a video signal, a region could pinpoint a screen location using X and Y coordinates.

Second, Atlas explicitly represents structural relationships by allowing annotations to name a set of 'children', without constraining how many 'parents' an annotation may have. The data storage format is an extension of the primary one employed by AGTK. The framework for defining the semantics of this relationship and for specifying which types of annotations expect which other types as children, MAIA, is still under development. It has the potential to be very flexible, especially if the semantics of the parent-child relationship can vary depending on the types of data objects that they link. The Atlas data model is implemented in Java as jAtlas (*jATLAS*, n.d.) and has been adopted as the basis of at least one annotation tool, Callisto (*Callisto*, n.d.). The developers plan both a query language and direct support for writing graphical user interfaces.

9. Discussion

The NITE XML Toolkit provides useful infrastructure for creating and working with language corpora, especially where higher-level annotations with complex structures are required, where data creation is distributed over several sites, and where the same basic material is to be annotated for a range of properties across different modalities. NXT is designed to support human users requiring access to the rich structure underlying linguistic annotation. NXT is not suitable for applications that require particularly fast processing or large amounts of data loaded at the same time, but it is perfectly possible to use a less expressive data model on data produced using NXT or stored in NXT format. Some users adopt NXT because they want to create annotations that are not supported in other tools, particularly for discourse markup such as dialogue acts, topic segmentation, and extractive summaries. Others up-translate existing annotations just for the ability to view and search them jointly, even

if their annotation is already complete. NXT has been used on a number of corpora well-known in the language engineering community, including the Switchboard Corpus (J. Carletta et al., 2004; Godfrey, Holliman, & McDaniel, 1992), the LeaP Corpus (Milde & Gut, 2002), and the ICSI Meeting Corpus (Carletta & Kilgour, 2005; Janin et al., 2003). Multimodal data sets are newer and therefore results for them are still in the pipeline, but there are at least five of them using NXT, the largest of which arises from a European collaboration with six sites contributing annotation and more using the data (Carletta et al., 2005). Less expected uses include an analysis of the text structure of Genesis in classical Hebrew and the coding of teaching strategies in tutorial discourse.

NXT is currently supported from several European and UK sources. The biggest and most important planned development, which is just starting, is to make queries run faster and to augment the query language with operators that have been requested by our users. The target new operators include basic arithmetic, the thirteen possible relationships between two time intervals (Allen, 1984), possibly extending them to structural order as well; and distance limitations on dominance and precedence. We expect to make these gains by building a query language implementation that translates queries into XQuery for execution using one of the many XQuery engines that has recently become available. If we build a successful implementation for static data, the easiest first step, we then plan to make our data model implementation and GUIs interoperable with it so that XQuery can be used as a back end for all NXT applications. Resource has also been committed over the next year for maintenance; rationalization of the existing documentation, which users report is adequate but is currently largely web-based; explicit description of the data paths for connecting NXT

to language processing tools such as part-of-speech taggers and machine learning; test suite improvements; memory management; and better up-translation of data to NXT format from channelTrans (*Extensions to Transcriber for Meeting Recorder Transcription*, n.d.), EventEditor (Sumec, n.d.), The Observer (Noldus, Trienes, Hendriksen, Jansen, & Jansen, 2000), TIGER (Tiger Project, n.d.), PRAAT (Boersma & Weenink, n.d.), ELAN (Max-Planck-Institute for Psycholinguistics, n.d.), and Eyelink II eyetrackers (SR Research, n.d.).

The NITE XML Toolkit is described more fully at http://www.ltg.ed.ac.uk/NITE/. NXT is open source software and as such is intended as a community resource. The libraries and working samples are available from Sourceforge (http://sourceforge.net/projects/nite/), as are active bug and feature request lists. This means that anyone can participate in NXT's future development, either by expressing their ideas for others to take up, by contributing code, or by investing resource earmarked against specific improvements.

10. Acknowledgements

NXT has been developed under European Commission funding (NITE, IST-2000-26095 and AMI, FP6-506811) and was based heavily upon the successes and failures of an earlier EC-funded prototype (MATE, LE4-8370). We are grateful for this funding and also for permission to use sample data from the EPSRC (UK) funded MONITOR project and from the PhD work of Hannele Nicholson for our worked example in section 7. We are also grateful to those who support Sourceforge and the Open Source Community.

Figure 1: An example of linguistic annotation for a short monologue exhibiting timing information, structural dominance requiring timing inheritance, relationships without timing implications, and a data type ontology.

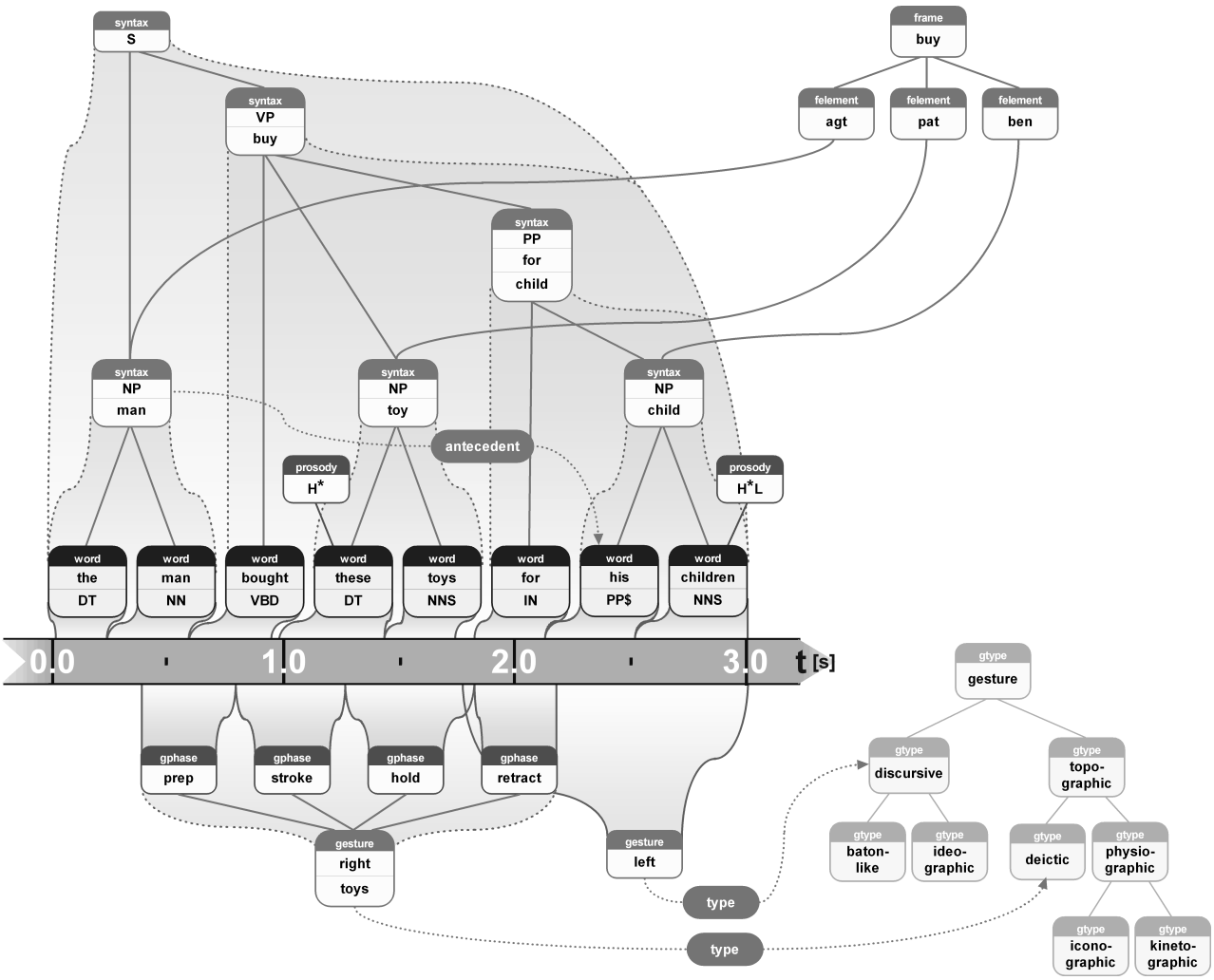
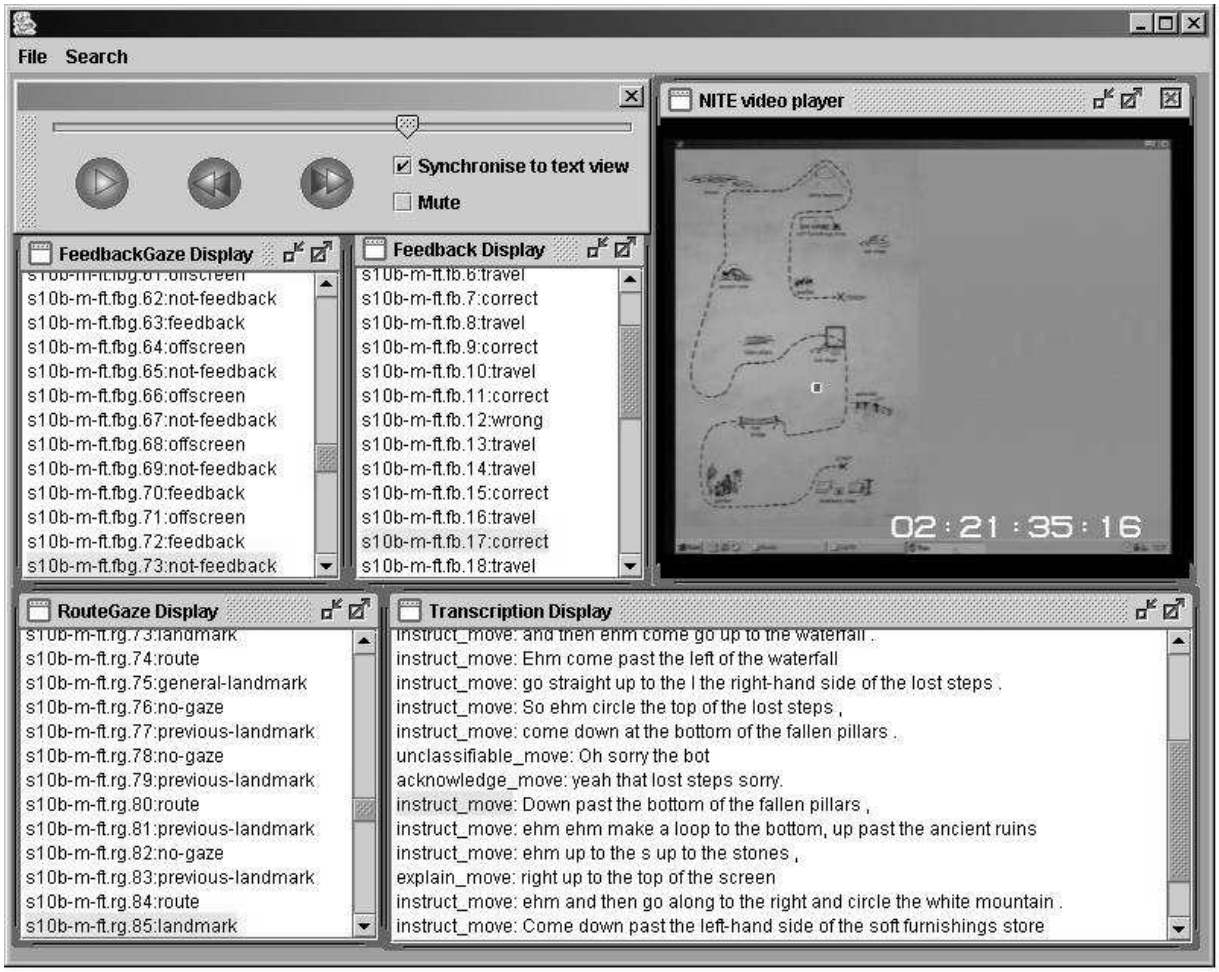


Figure 2: A screenshot of the MONITOR data display.



References

AGTK: Annotation Graph Toolkit. (n.d.). Retrieved 06 Jan, 2005, from http://agtk.sourceforge.net/.

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence, 23*(2), 123-154.

Anderson, A. H., Bader, M., Bard, E. G., Boyle, E., Doherty, G., Garrod, S., et al. (1991). The HCRC Map Task Corpus. *Language and Speech, 34*(4), 351-366.

ATLAS Project. (2000, 6 Feb 2003). Retrieved 1 Mar, 2004, from http://www.nist.gov/speech/atlas/.

Bird, S., & Liberman, M. (2001). A Formal Framework for Linguistic Annotation. *Speech Communication, 33*(1-2), 23-60.

Boag, S., Chamberlin, D., Fernandez, M. F., Florescu, D., Robie, J., Siméon, J., et al. (n.d., 29 October 2004). *XQuery 1.0: An XML Query Language*. Retrieved 06 Jan, 2005, from http://www.w3.org/TR/xquery/.

Boersma, P., & Weenink, D. (n.d.). *Praat: doing phonetics by computer*. Retrieved 14 Oct, 2005, from http://www.fon.hum.uva.nl/praat/.

Callisto. (n.d., 20 April 2004). Retrieved 06 Jan, 2005, from http://callisto.mitre.org/

Carletta, J., Ashby, S., Bourban, S., Flynn, M., Guillemot, M., Hain, T., et al. (2005, 11-13 July). *The AMI Meeting Corpus: A Pre-Announcement.* Paper presented at the 2nd Joint Workshop on Multimodal Interaction and Related Machine Learning Algorithms, Edinburgh.

Carletta, J., Dingare, S., Nissim, M., & Nikitina, T. (2004, May 26-28). *Using the NITE XML Toolkit on the Switchboard Corpus to study syntactic choice: a case study.* Paper presented at the Fourth Language Resources and Evaluation Conference, Lisbon, Portugal.

Carletta, J., Kilgour, J., O'Donnell, T., Evert, S., & Voormann, H. (2003). The NITE Object Model Library for Handling Structured Linguistic Annotation on Multimodal Data Sets. In N. Ide (Ed.), *Proceedings of the EACL Workshop on Language Technology and the Semantic Web (3rd Workshop on NLP and XML, NLPXML-2003)*.

Carletta, J. C., Isard, A., Isard, S., Kowtko, J., Doherty-Sneddon, G., & Anderson, A. (1997). The reliability of a dialogue structure coding scheme. *Computational Linguistics, 23*(1), 13-31.

Carletta, J. C., & Kilgour, J. (2005). The NITE XML Toolkit Meets the ICSI Meeting Corpus: Import, Annotation, and Browsing. In S. Bengio & H. Bourlard (Eds.), *Machine Learning for Multimodal Interaction: First International Workshop, MLMI 2004, Martigny, Switzerland, June 21-23, 2004, Revised Selected Papers* (pp. 111-121). Berlin: Springer-Verlag.

Carletta, J. C., McKelvie, D., Isard, A., Mengel, A., Klein, M., & Møller, M. B. (2004). A generic approach to software support for linguistic annotation using XML. In G. Sampson & D. McCarthy (Eds.), *Corpus Linguistics: Readings in a Widening Discipline*. London and NY: Continuum International.

Cassidy, S., & Harrington, J. (2001). Multi-level annotation in the Emu speech database management system. *Speech Communication, 33*, 61-77.

Clark, J., & DeRose, S. (1999, 16 November). *XML Path Language (XPath) Version 1.0*. Retrieved 26 May, 2003, from http://www.w3.org/TR/xpath.

DeRose, S., Maler, E., & Daniel Jr., R. (2001, 11 September). *XML Pointer Language (XPointer) Version 1.0*, from http://www.w3.org/TR/xptr/.

DeRose, S., Maler, E., & Orchard, D. (2001, 27 June). *XML Linking Language (XLink) Version 1.0*. Retrieved 05 April, 2004, from http://www.w3.org/TR/xlink/.

The Edinburgh Speech Tools Library. (n.d.). Retrieved 23 Dec, 2004, from http://www.cstr.ed.ac.uk/projects/speech_tools/.

The EMU Speech Database System. (2001). Retrieved 23 Dec, 2004, from http://emu.sourceforge.net/.

Evert, S., Carletta, J. C., O'Donnell, T. J., Kilgour, J., Vogele, A., & Voormann, H. (2002). *NXT Data Model*. Retrieved 26 May, 2003, from http://www.ltg.ed.ac.uk/NITE/.

Extensions to Transcriber for Meeting Recorder Transcription. (n.d.). Retrieved 14 October, 2005, from http://www.icsi.berkeley.edu/Speech/mr/channeltrans.html.

Godfrey, J., Holliman, E., & McDaniel, J. (1992, March). *Switchboard: Telephone speech corpus for research development.* Paper presented at the International Conference on Acoustics, Speech, and Signal Processing, San Francisco, CA, USA.

Heid, U., Voormann, H., Milde, J.-T., Gut, U., Erk, K., & Padó, S. (2004). *Querying both time-aligned and hierarchical corpora with NXT Search.* Paper presented at the Fourth Language Resources and Evaluation Conference, Lisbon, Portugal.

ISO/TC37/SC4. (n.d.). *Welcome to ISO/TC 37/SC 4 Language Resources Management*. Retrieved 17 Oct, 2005, from http://www.tc37sc4.org/index.php.

Janin, A., Baron, D., Edwards, J., Ellis, D., Gelbart, D., Morgan, N., et al. (2003). *The ICSI Meeting Corpus.* Paper presented at the IEEE International Conference on Acoustics, Speech and Signal Processing, Hong Kong.

jATLAS. (n.d., 15 October 2003). Retrieved 06 Jan, 2005, from http://jatlas.sourceforge.net/.

Laprun, C., Fiscus, J. G., Garofolo, J., & Pajot, S. (2002, May). *A Practical Introduction to ATLAS.* Paper presented at the 3rd International Conference on Language Resources and Evaluation (LREC), Las Palmas.

Lickley, R. J. (1998). *HCRC Disfluency Coding Manual* (HCRC Technical Report No. 100): Human Communication Research Centre, University of Edinburgh.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., et al. (1994). *The Penn Treebank: Annotating predicate argument structure.* Paper presented at the ARPA Human Language Technology Workshop.

Max-Planck-Institute for Psycholinguistics. (n.d.). *Tools*. Retrieved 14 Oct, 2005, from http://www.mpi.nl/tools/elan.html

Milde, J.-T., & Gut, U. (2002, 11-13 April 2002). *A prosodic corpus of non-native speech.* Paper presented at the Speech Prosody 2002 conference, Aix-en-Provence.

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.

NXT Search. (n.d., 3 July 2003). Retrieved 1 Mar, 2004, from http://www.ims.uni-stuttgart.de/projekte/nite/.

SR Research. (n.d.). *Complete EyeTracking Solutions*. Retrieved 14 Oct, 2005, from http://www.eyelinkinfo.com/.

Strassel, S., Maxwell, M., & Cieri, C. (2003). Linguistic Resource Creation for Research and Technology Development: A Recent Experiment. *ACM Transactions on Asian Language Information Processing, 2*(2), 101-117.

Sumec, S. (n.d.). *Event Editor*. Retrieved 14 Oct, 2005, from http://www.fit.vutbr.cz/research/grants/m4/editor/index.htm.cs.iso-8859-2

Taylor, P., Black, A., & Caley, R. (2001). Heterogeneous Relation Graphs as a Formalism for Representing Linguistic Information. *Speech Communication, 33*(1-2), 153-174.

TEI Consortium. (1999). *TEI: The Text Encoding Initiative*. Retrieved 26 May, 2003, from http://www.tei-c.org/.

The Apache XML Project. (n.d.). *Xerces2 Java Parser Readme*. Retrieved 15 March, 2004, from http://xml.apache.org/xerces2-j/index.html.

The Language Technology Group. (n.d.). *LTG Software*. Retrieved 15 March, 2004, from http://www.ltg.ed.ac.uk/software/index.html.

Tiger Project. (n.d., 17 Nov 03). *Linguistic Interpretation of a German Corpus*. Retrieved 1 Mar, 2004, from http://www.ims.uni-stuttgart.de/projekte/TIGER/

W3C. (2005, 16 September). *EMMA: Extensible MultiModal Annotation markup language*. Retrieved 17 Oct, 2005, from http://www.w3.org/TR/2005/WD-emma-20050916/.

Wittenburg, P., Broeder, D., & Sloman, B. (2000). *EAGLES/ISLE: A Proposal for a Meta Description Standard for Language Resources, White Paper.* Paper presented at the LREC 2000 Workshop, Athens, Greece.