# The NITE Object Model Library for Handling Structured Linguistic Annotation on Multimodal Data Sets

# The NITE Object Model Library for Handling Structured Linguistic Annotation on Multimodal Data Sets

**Jean Carletta, Jonathan Kilgour, and Tim O'Donnell**
School of Informatics
University of Edinburgh
J.Carletta@ed.ac.uk
jonathan@inf.ed.ac.uk
timo@inf.ed.ac.uk

**Stefan Evert and Holger Voormann**
Institut für Maschinelle Sprachverarbeitung
Universität Stuttgart
evert@IMS.Uni-Stuttgart.DE
voormann@IMS.Uni-Stuttgart.DE

## Abstract

The NITE Object Model Library is an implemented set of routines for loading, accessing, manipulating, and serializing linguistic data. It is similar in spirit to the data handling provided by the Annotation Graph Toolkit, but is aimed at data that is heavily cross-annotated with structured information, and thus chooses higher expressivity at the cost of processing speed. We describe our open-source implementation and the XML-based data storage format that it assumes, and discuss the circumstances under which it is a useful addition to previous data handling techniques.

## 1 Introduction

Linguistic corpora are central to natural language processing, especially given the current predominance of statistical methods. Increasingly, because of the growing interest in multimodal interfaces, corpora include interactions (either human-human or human-computer) where all modalities are captured, with multiple, synchronized audio and video recordings and interface traces. Understanding human communication requires all of the modalities to be considered together, since gesture, speech, body movement, and so on combine to produce the message. Although historically computational linguists have usually restricted themselves to one or two kinds of annotation — syntax, say, or dialogue acts — they now require mechanisms by which they can add many different kinds of annotation to the same basic data and relate them together.

The NITE Project (http://www.nis.sdu.dk) is funded by the European Commission to provide infrastructural technology for working with heavily cross-annotated multimodal data sets. Among other efforts, we have characterized the types of structures inherent in the annotations needed to study multimodal interaction, defined a model of the building blocks and relationships among them needed to represent these structures, and constructed a library of routines for reading, accessing, manipulating, and serializing linguistic data according to the chosen model. This effort shares much in common with both the Annotation Graph Toolkit (Ma, Lee, Bird, & Maeda, 2002) and with ATLAS (Laprun, Fiscus, Garofolo, & Pajot, 2002). However, in keeping with the aim of supporting work with heavily cross-annotated data sets, our model allows easier access to rich structural information about the data than these other systems.

## 2 What sort of model?

This work is motivated by the sorts of data modelling concerns that are raised by having

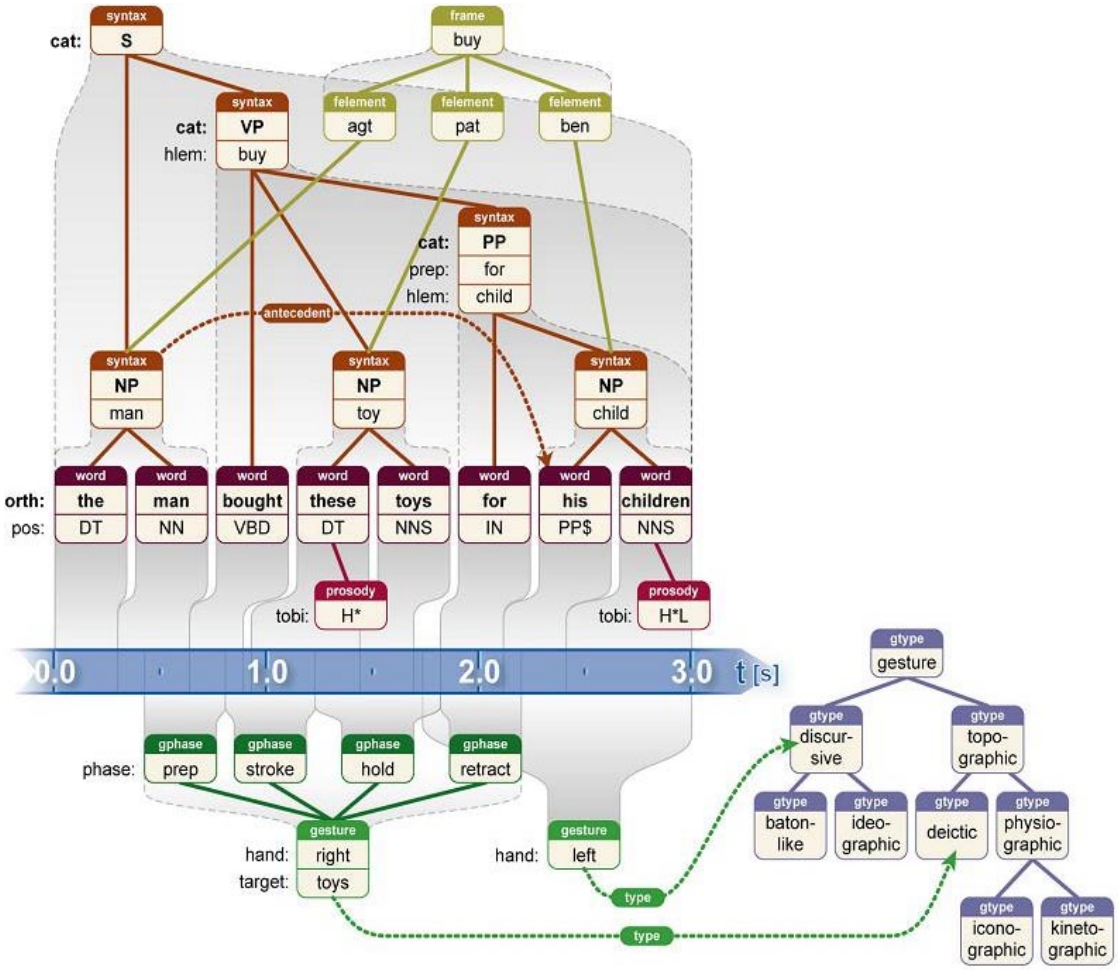


Figure 1: An example of linguistic annotation for a short monologue exhibiting timing information, structural dominance requiring timing inheritance, relationships without timing implications, and a data type ontology.

many kinds of annotation, for linguistic levels ranging from phonology to pragmatics, on the same basic speech or language material. 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. The HCRC Map Task (Anderson et al., 1991) is one example of a corpus that has been prepared to answer these questions, with annotations (http://www.hcrc.ed.ac.uk/maptask) that range from orthography and syntax to reference and dialogue structure.

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). Figure 1 gives a toy example that is engineered to show this full range of data relationships.

## 3 The NITE Object Model

It is clear from our requirements that although there are certain general properties to which sets of linguistic annotations adhere, data set designers need the freedom to specify the exact structure required for any particular data set. Because of this, we specify data handling in terms of an object model that defines the building blocks that together make up an annotation set. This is the same approach that is taken, for instance, in XML processing for the Document Object Model (DOM) that defines a tree-structured document structure in terms of a set of nodes, where pairs of nodes can be related to each other by structural dominance.

### 3.1 The graph structure

The NITE Object Model consists of a general graph structure, and then some properties imposed on top of that graph structure that make using that structure more computationally tractable whilst still expressing the sorts of relationships that are prevalent among annotations.

The NITE Object Model is a graph where the nodes are required to have a simple type and may additionally have attribute-value pairs elaborating on the simple type, timings, children that are structurally dominated, and relations that fulfill a named role. These nodes are called elements.

The simple type is a string.

An attribute is identified by a simple label string and takes a value that conforms to one of three types: a string, a number, or an enumeration. The simple type of the element determines what attributes it can contain. For any element, the simple type plus the attribute-value pairs defined for the element represent its type.

Timing information can be present, and is represented by reserved start and end attributes containing numbers that represent offsets from the start of the synchronized signals.

The children are represented by an (ordered) list of other elements.

The features are represented by a list of role and filler pairs. A role is a simple label string that has an expected arity, or number of elements expected to fill the role: one, or one-or-more.

### 3.2 Additional properties

The object model also imposes some properties on this general graph structure to do with orderings. Firstly, this graph must be acyclic so that its transitive closure can be interpreted as a dominance relation. Secondly, there must not be more than one path between any two elements. Because of these constraints, the parent-child graph (which, unlike a tree, allows children to have multiple parents) decomposes into a collection of intersecting tree-like structures, called hierarchies. Each hierarchy has its own sequential ordering (similar to an ordered tree), but these orderings must be consistent where hierarchies intersect.

If an element has timing information, the element's start time must be less than or equal to its end time. In addition, if elements in a dominance relation both have timing information, the time interval associated with the ancestor must include that of the descendant. The times of elements need not be consistent with any of the sequential orderings. Timing information can thus be used to define an additional partial ordering called temporal precedence, which is not restricted to a single hierarchy.

Note that featural relations are exempt from any structural or timing constraints. They are useful for providing generalized attributes that are filled by elements instead of strings and for defining additional arbitrary graph structures overlaying the main parent-child graph.

It is only possible in a paper of this length to provide an informal gloss of the NITE Object

Model. The model is formally defined in (Evert et al., 2002).

## 4 The NITE Data Set Model

Our object model is simply an abstract graph structure with a number of properties enforced on it that govern orderings. However, it can be difficult for data set designers to think of their data in terms this abstract, rather than the more usual concepts such as corpus, signal, and annotation. In addition, for practical reasons it can be useful to require data represented in such a model not to contain cycles, at least in the child relationships. For this reason, we provide a data set model in these familiar terms that can easily be expressed using our object model and from whose structure the essential properties we require regarding orderings and acyclicity fall out. Data set designers can use this level of the model to describe their designs, and by providing metadata that expresses the design formally, make it possible to validate the overall structure of any specific data set against their intended design.

Here we describe the main entities and relationships that occur in our data set model.

Observation. An observation is the data collected for one interaction — one dialogue or small group discussion, for example.

Corpus. A corpus is a set of observations that have the same basic structure and together are designed to address some research need. For each simple data type, metadata for the corpus determines what attribute-value pairs can be used to refine the type, whether or not elements of that type have timing information and/or children, and what features can be present for them.

Agent. An agent is one interactant in an observation. Agents can be human or artificial. We provide the concept of agent so that annotations can be identified as describing the behaviour of a single agent or of the interacting group as a whole. We do not provide a way of identifying other subsets of agents acting together as a group.

Signal. A signal is the output from one sensor used to record an observation: for example, an audio or video file or blood pressure data. An observation may be recorded using more than one signal, but these are assumed to be synchronized, so that timestamps refer to the same time on all of them.

Annotation. An annotation is an element that describes part of an observation. When an element is used to represent an annotation, it will have a data type and may have timing information, features and children. When an annotation has children, it means that the parent annotation dominates the children (for instance, in the relationships between words and syllables).

Object and Object Set. An object is an element that represents something in the universe to which an annotation might wish to point. An object might be used, for instance, to represent the referent of a referring expression or the lexical entry corresponding to a word token spoken by one of the agents. When an element is used to represent an object, it will have a data type and may have features, but no timing or children. An object set is a set of objects of the same or related data types. Object sets have no inherent order.

Complex type and Ontology. An ontology is a tree of elements that makes use of the parent/child structure to specify specializations of a data type. In the tree, the root is an element naming some simple data type that is used by some annotations. In an ontology, if one type is a child of another, that means that the former is a specialization of the latter. The children of an element in an ontology are unordered and can define attributes specific to that specialization. We have defined ontologies to make it simpler to assign a basic type to an annotation in the first instance, later refining the type. When this is done, the annotation will retain its basic type but can use a feature with the special reserved role “type” of arity one to point to the appropriate specialization in the corresponding ontology.

Layer. A layer is a set of annotations that together span an observation in some way, containing all of the annotations for a

particular agent or for the interaction that are either of the same type or drawn from a set of related types. Which data types belong together in a layer is defined by the corpus metadata. For instance, the TEI defines a set of tags for representing words, silences, noises, and a few other phenomena, which together span a text and make up the orthographic transcription. In this treatment, these tags would form a layer in our data set model.

Time-Aligned Layer. A time-aligned layer is a layer where the annotations conform to data types that have timing information. By having this structure, these annotations refer directly to signal.

Structural Layer. A structural layer is a layer where the annotations conform to data types that can have children. The children of a structural layer are constrained to be drawn from a single layer, which, in order to allow recursive structures, can be itself.

Featural Layer. A featural layer is a layer where the annotations conform to data types that may have features, but not children or timing information. Features with the same role in the same featural layer are constrained to be filled by elements drawn from a single layer. Annotations in a featural layer may not be children of any other annotations. A featural layer draws together other annotations into clusters that represent phenomena that do not adhere to our timing relationships. For instance, a featural layer might contain annotations that pair deictic gestures with deictic pronouns. Since deictic pronouns and their accompanying gestures can lag each other by arbitrary amounts, there is no sense in which the deictic pair spans from the start of one to the end of the other.

Coding. A coding is a sequence of one or more layers, all either for the same agent or for the interaction as a whole, where each layer's children are taken from the next layer in the sequence, ending either in a layer with no children, in a layer whose children are in the top layer of another coding. Codings defined in this way consist of tree structures, and the relations among codings allow for the sort of multiply rooted tree illustrated in figure one. Featural layers occur in single-layer codings of their own. Since most coherent sets of codes applied to the data at the same time fit into tree structures, for many corpora, the codings will correspond to what can be loosely thought of as types of annotation.

Together, these definitions preserve the ordering properties that we desire; intuitively, time-aligned and structural layers are ordered, and timings can percolate up structural layers from a time-aligned layer at the base. The layer structure within a coding prohibits cycles. Different time-aligned layers can exist for the same signal, with different annotation layers anchored to each, and featural layers gathering phenomena from each into sets.

## 5 Data and metadata storage

It is not possible to define a data handling library without first defining the storage format that the loading routines expect to encounter and the serialization routines expect to write. We have chosen a format that separates each coding for each observation into a separate file. For agent codings, there is one file per agent; for interaction codings, there is one file that covers all of the agents together. Each object set and ontology occupies one file that can be referenced from codings relating to any observation in the corpus.

The files themselves are given in an XML format. By definition, each coding may only contain hierarchically decomposable layers of elements. The files therefore contain a <root> element at the document level that identifies the coding within, and then a tree structure document where nodes at the first level correspond to the first layer tags, nodes at the second level correspond to the second layer tags, and so on down to the leaves of the coding. Tags are named directly by the types given in the metadata definition, with the attributes defined. Links to other codings, which can occur anywhere in features but only at the tree's leaves as children, are specified using either XPointer/XLink syntax or an older syntax used by LT-XML's stand-off facilities (McKelvie, Brew, & Thompson, 1998). Metadata describing the data associated with a

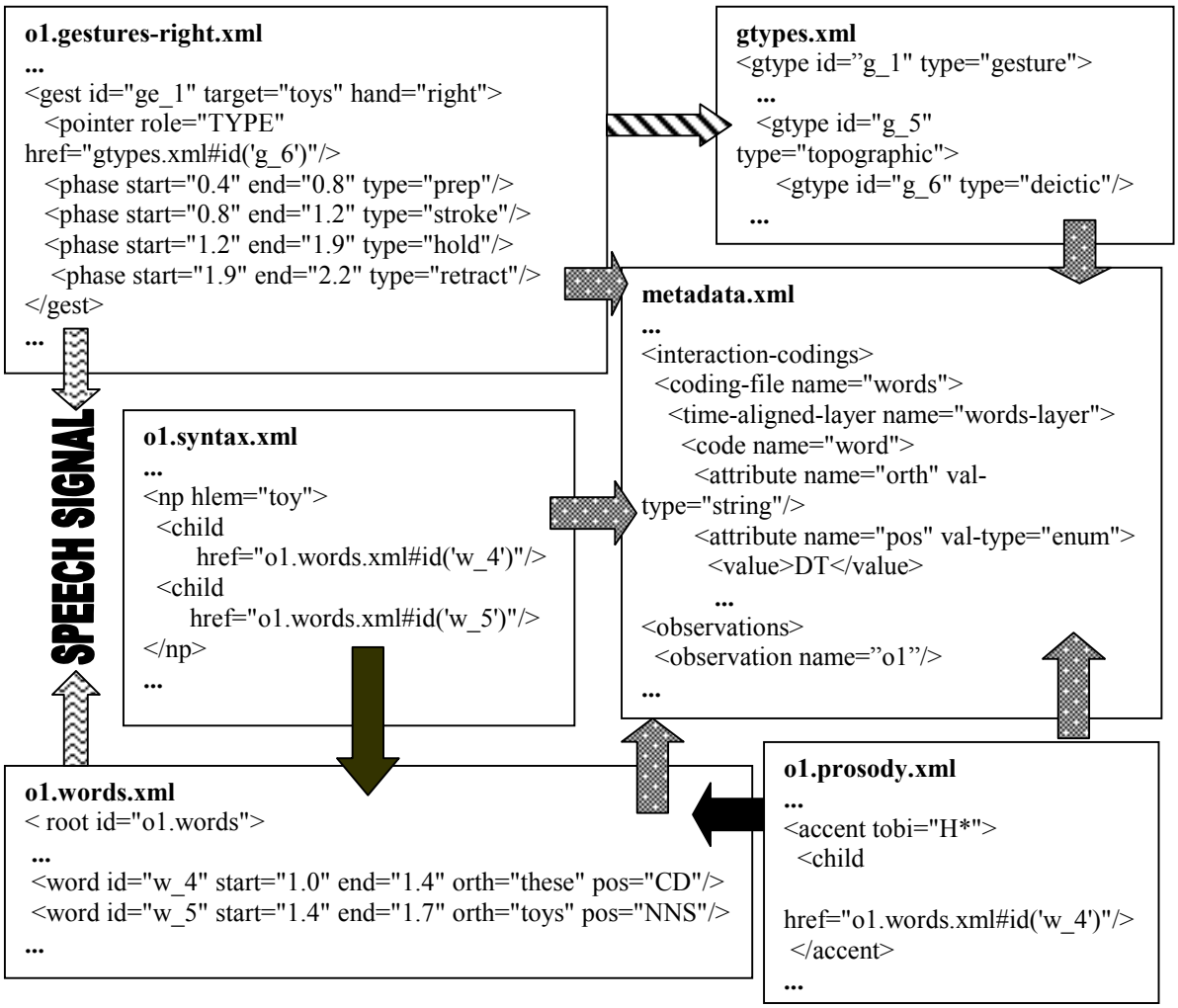


Figure 2: The XML data and metadata storage for part of the example in figure 1.

corpus in terms of the data set model is also expressed in a simple, fixed-schema XML format. Figure 2 gives a sense of what the data and metadata look like for part of figure 1. In the figure, black arrows show links between annotations, striped arrows show links from annotations to type hierarchies, freckly arrows show links between data and the metadata defining it, and wavy arrows show links from annotations to signal. The figure uses LT-XML format links and omits references to the "nite" namespace and all ids except those used for linking, in the interests of space.

This choice of format has a number of advantages over the other possibilities:

the data format is inspectable and amenable to mainstream XML processing techniques, such as stylesheets, even without our handling library;

it can be validated using information expressed in any of the standard XML schema languages, including, say, schemas or DTDs generated from the metadata file;

by compartmentalizing data of different types, part or all of the data can be loaded for different purposes, and different people can change different parts of the data **at the same time**;

the storage format directly exposes tree structures underlying the annotation, and thus to the extent that annotations are tree-structured, makes them easier to work with

than, for instance, data expressed in Atlas Interchange Format.

## 6 The NOM Library

The NOM library, the first release of which is available from http://www.ltg.ed.ac.uk/NITE, provides an API for loading, saving and manipulating both the data and metadata associated with a corpus. The library is implemented in Java, though can of course be called from other programming languages too.

The NOM library design explicitly caters for a split between read and write functionality. For applications like corpus search, we may be able to make significant efficiency gains by making the assumption that the corpus is not changing 'under our feet'. Our first NOM implementation provides both read and write functionality, so does not assume the corpus is static.

### 6.1 Metadata handling

Metadata is essential to operations like loading and saving data as it defines both the expected structure of an annotated corpus and where to find the signals and data files on disk. In our implementation, the metadata is tightly coupled to the object model library, which means that it can only handle data that conforms to our data set model even though the object model itself is less restrictive. This is because the possible advantages of separating the NOM from the data set model are outweighed by the efficiency gains we can achieve through a tight coupling here, and because most applications will wish to validate against the data set model anyway.

The metadata API provides access to information about the corpus described in the terms of the data set model. It also provides some limited routines for common metadata manipulations that we expect to be required programmatically, such as changing the paths to the data files. We expect more substantive corpus design changes to be made using an XML editor.

### 6.2 Data loading and serialization

Once the metadata for a corpus has been loaded, data can be loaded using our API either for an entire corpus or for one observation at a time. In the latter case, more observations can be loaded to build up the corpus incrementally.

Serialization methods save either the entire loaded corpus or just the files that have changed. The implementation offers a choice of link style between XPointer/XLink and LT-XML.

### 6.3 Data handling

We provide an iterator that visits each node in the object model exactly once, as well as access by element type or id. Elements are linked to their specifications in the metadata, so that the information required to check attribute and structural constraints is readily accessible. It is possible to navigate from an element to its relatives in the graph using the API. As one would expect, the API also provides routines for manipulating the object model, for instance, by adding an attribute, child, or feature to an existing element, or adding a new element to a time-aligned layer with a particular start and end time. Whenever elements are added or deleted, or their start and end times are changed, the NOM remains internally coherent by percolating the time changes up through any dominating structural layers. This percolation assumes that for any two adjacent children, the end time of the first is before the start time of the second, and that parents take their start time from their first child and end time from their last one..

### 6.4 Data sharing

The NOM library also provides facilities for sharing a single in-memory data set between multiple applications. In order to do this, we introduce the concept of a NOM controller and a NOM view. Any views that are registered with the controller will be informed of edits to the data, and will have any edits of its own passed along to the other registered views. There are two different granularities at which changes can be viewed. A view can simply ask to be informed each time any change is made to the data, which would allow one to re-generate an entire data display with the newly edited data. Alternatively a view can receive much more detailed information about edits that are intended to enable incremental display changes.

## 7 Discussion

We provide an object model for representing linguistic annotation that is oriented towards heavily cross-annotated data sets, and a data set model in terms of concepts that data set designers will be able to use that naturally enforce the structural properties that the object model requires. Our implementation combining these models can be used to handle our target data sets, if they are stored in a particular stand-off XML data format. Our system differs from other libraries for working with annotated corpora, as befits our development aims. The object model is quite similar to ATLAS in its dominance structures, but adds featural links that do not entail timing relationships, complex data types, and objects, and is less flexible about how it anchors annotations in signal. Our data set model is more prescriptive than MAIA, the metadata proposed for ATLAS, which is its closest equivalent.

In constructing a model that highlights the relationships among annotations in a cross-annotated data set, we do not suggest that this model is the best one for all purposes. Our model simply gives one way of handling a data set that brings these relationships to the fore. Where a data set only has one type of annotation, or where annotations have no inherent structure behind their timings, using such a model only adds overhead processing costs. Similarly, even when a data set is amenable to treatment with this model, there will be processes for which a simpler model is more appropriate. For instance, when calculating n-grams, a model such as the annotation graph that exposes labels and timing information will be more efficient.

## 8 Future Work

We are currently considering how to improve processing efficiency for our implementation, whether or not to provide a read-only version, and what validation to provide. The NOM provides some validation during processing that can be turned off to increase speed, but it may also be useful to generate XML Schemas from the metadata that allow at least the structural and typing constraints to be checked off-line.

In addition to the NOM, we plan other software as infrastructural support. The primary purpose of building any data model is, of course, to expose the data set to query. In addition to the work described here, we have also designed a query language (Evert & Voormann, 2002) that makes use of the inherent structure of this model in order to allow the easy expression of queries that relate different annotations together. We are also updating an idea prototyped in the MATE project (McKelvie et al., 2001) for what is now standard XML technology. This involves implementing a library of Java display objects that can be called upon for writing data displays and interfaces, and an engine for constructing tailored coding interfaces from a stylesheet that declaratively specifies the interface's appearance and behaviour.

## References


Anderson, A. H., Bader, M., Bard, E. G., Boyle, E., Doherty, G., Garrod, S., Isard, S., Kowtko, J., McAllister, J., Miller, J., Sotillo, C., Thompson, H., & Weinert, R. (1991). The HCRC Map Task Corpus. *Language and Speech, 34*(4), 351-366.

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

Evert, S., & Voormann, H. (2002). *NITE Query Language*. From http://www.ltg.ed.ac.uk/NITE/documents.html.

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.

Ma, X., Lee, H., Bird, S., & Maeda, K. (2002). *Models and Tools for Collaborative Annotation.* Paper presented at the Third International Conference on Language Resources and Evaluation.

McKelvie, D., Brew, C., & Thompson, H. (1998). Using SGML as a Basis for Data-Intensive Natural Language Processing. *Computers and the Humanities, 31*(5), 367-388.

McKelvie, D., Isard, A., Mengel, A., Møller, M. B., Grosse, M., & Klein, M. (2001). The MATE Workbench - an annotation tool for XML coded speech corpora. *Speech Communication, 33*(1-2), 97-112.