Behavior Research Methods, Instruments, & Computers
2003, 35 (3), 353-363

The NITE XML Toolkit: Flexible annotation for multimodal language data

JEAN CARLETTA
University of Edinburgh, Edinburgh, Scotland

STEFAN EVERT and ULRICH HEID
Universität Stuttgart, Stuttgart, Germany

JONATHAN KILGOUR and JUDY ROBERTSON
University of Edinburgh, Edinburgh, Scotland

and

HOLGER VOORMANN
Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Stuttgart, Germany

Multimodal corpora that show humans interacting via language are now relatively easy to collect. Current tools allow one either to apply sets of time-stamped codes to the data and consider their timing and sequencing or to describe some specific linguistic structure that is present in the data, built over the top of some form of transcription. To further our understanding of human communication, the research community needs code sets with both timings and structure, designed flexibly to address the research questions at hand. The NITE XML Toolkit offers library support that software developers can call upon when writing tools for such code sets and, thus, enables richer analyses than have previously been possible. It includes data handling, a query language containing both structural and temporal constructs, components that can be used to build graphical interfaces, sample programs that demonstrate how to use the libraries, a tool for running queries, and an experimental engine that builds interfaces on the basis of declarative specifications.

Human interaction research has always been inventive in its use of the latest technology. Even 50 years ago, Bales (1951) adopted one-way mirrors to observe his groups and then had to design motorized paper scrolls so that his observers could keep up during live scoring. Since then, signal recording technologies have advanced significantly; video cameras are portable, microphones can be arranged to pick up individual subjects even without the use of wires, and multiple signals can be synchronized using a mixing desk. Not only that, but now that every garage band makes music videos, these technologies are so cheap that researchers can focus less on cost and more on what would make for their ideal data capture. With these advances in signal recording come new ideas about what sort of data to collect and how to use them.

One research area that can benefit greatly from better signal recording is the study of how people use language. When people communicate, gestures, postural shifts, facial expressions, backchannel continuers such as "mm-hmm," and spoken turns from the subjects all work in concert to bring about mutual understanding (Goodwin, 1981). Apart from the scientific good of understanding how this process works, information about it is in demand for applications ranging from the documentation of endangered languages to animation for computer games. Observational analysis packages can help us determine some things about the timing, frequency, and sequencing of communicative behaviors, but that is not enough. In language data, behaviors are related less by their timing than by their structure: Pronouns have discourse referents, answers relate to questions, and deictic instances of the word "that" are resolved by pointing gestures that themselves relate to real-world objects, but with no guarantees about *when* the related behavior will occur. Linguistic analysis reveals this structure, but current tools only support specific codes and structures and only allow them to be imposed over the top of a textual transcription. This approach discards temporal information and makes it difficult to describe behaviors from different subjects that happen at the same time.

We gratefully acknowledge support of the NITE project by the European Commission's Human Language Technologies Programme. The samples described in the paper use data kindly provided either to us personally or to the community at large by the Smartkom project (http://smartkom.dfki.de/), by ISIP's Switchboard project (http://www.isip.msstate.edu/projects/switchboard/), and by the University of Edinburgh's Human Communication Research Centre (http://www.hcrc.ed.ac.uk/). The software described in this paper is available for download from http://www.ltg.ed.ac.uk/NITE. Correspondence concerning this article should be addressed to J. Carletta, Human Communication Research Centre and Language Technology Group, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9LW, Scotland (e-mail: j.carletta@edinburgh.ac.uk).

Signal recording has advanced enough that human language data can be fully documented, but data recording and analysis still fall short.

The NITE XML Toolkit is free open-source software that addresses this deficiency by supporting the development of tools for the creation and analysis of codes that can have both a temporal relationship to a set of synchronized signals and structural relationships to other codes.

Motivation: What Sort of Data?

Since the sort of data and analysis our toolkit supports is very different from what other tools currently afford, perhaps the best way to explain what the toolkit enables is by example. Figure 1, which is artificially constructed for simplicity, contains a spoken sentence that has been coded with fairly standard linguistic information, shown above the representation of the timeline, and with gestural information, shown below it. The lowest full layer of linguistic information is an orthographic transcription consisting of words marked with part-of-speech tags (in this set, the tag "PP$" stands for "personal pronoun"). Some words have some limited prosodic information associated with them in the form of pitch accents, designated by their TOBI codes (Silverman et al., 1992). Building upon the words is a syntactic structure—in this formalism, a tree—with a category giving the type of syntactic constituent (sentence, noun phrase, verb phrase, and so on) and the lemma, or root form, of the word that is that constituent's head. Prepositional phrases, or PPs, additionally specify the preposition type. The syntactic constituents are not directly aligned to signal, but they inherit timing information from the words below them. The very same syntactic constituents slot into a semantic structure that describes the meaning of the utterance in terms of a semantic frame (in this case, a buying event) and the elements that fill the roles in the frame (the agent, the patient, and the beneficiary). The last piece of linguistic information, a link between the syntactic constituent "the man" and the personal pronoun "his," shows that the former is the antecedent of the latter in a coreference relationship.

Meanwhile, the gesture coding shows two timed gestures and their relationship to a static gesture ontology. In the ontology, one type is below another if the former is a subtype of the latter. The first gesture, with the right hand, is

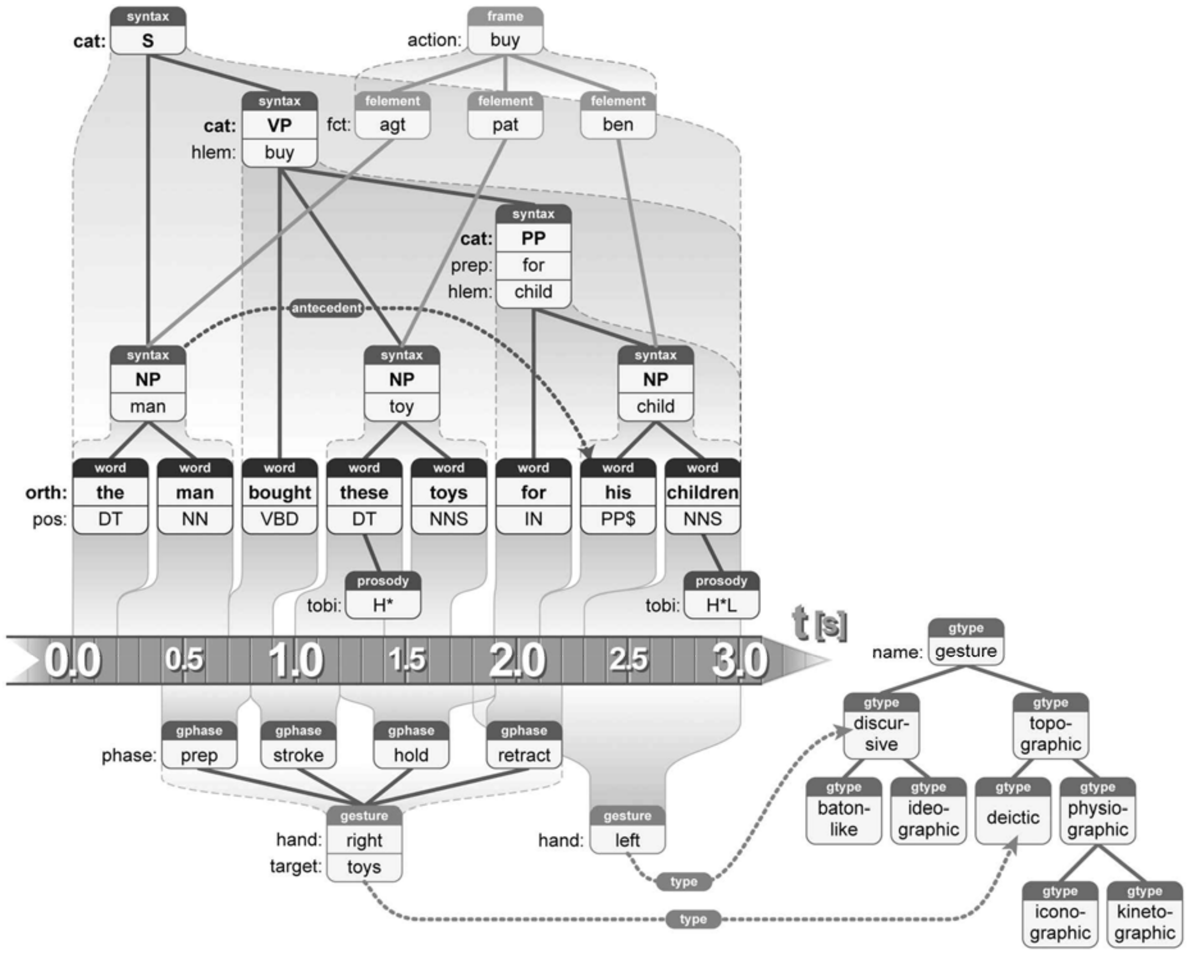


Figure 1. An example human utterance coded on several modalities, including detailed language coding.

a deictic, or pointing, gesture where the target of the pointing is some toys. This gesture is divided into the usual phases of preparation, stroke, hold, and retraction. The second gesture, made with the left hand, is discursive, but the coder has chosen not to qualify this type further. Gesture types could be represented for the gestures directly, in the same way that parts of speech are represented for words. However, linking into an ontology has the advantage of making clear the hierarchical nature of the gesture tag set.

All of these kinds of information are used frequently within their individual research communities. No previous software allows them to be integrated in a way that expresses fully how they are related and makes the relationships easy to access. And yet this integration is exactly what is required in order to understand the full import of this communicative act. The same requirements are in force for empirical work on a wide range of issues concerning language use: Speaker pauses and hearer backchannel continuers are thought to occur at major syntactic boundaries, an argument builds up using rhetorical relations that together span a text, postural shifts may signal a desire to take a speaking turn, and so on. These kinds of investigations are what the NITE XML Toolkit supports.

Motivation: What Sort of Support?

There are a great many existing tools that do some of what users of this new sort of data need, and it would be foolish to try to replace them. Transcriber (Barras, Geoffrois, Wu, & Liberman, 2001) is well known for transcription, and there are many different tools for adding linguistic annotations with specific structures to data, both by hand and automatically (e.g., The Alembic Workbench, Day et al., 1997, for tagging text corpora by hand; LT-POS, Language Technology Group, n.d., an automatic part-of-speech tagger). Several tools (e.g., Anvil, Kipp, 2001; TASX, Milde & Gut, 2001; The Observer, Noldus, Trienes, Hendriksen, Jansen, & Jansen, 2000) support the creation of time-stamped state and event codes, either in independent layers or in layers related in a hierarchical decomposition. There are also several statistical analysis packages in widespread use. Most of these packages store their data in XML, have XML import and/or export, or are being considered for XML.[1] XML (World Wide Web Consortium, n.d.–a) is a standard way of defining formats for and storing semistructured data and is popular in commercial applications, such as product databases. Transformation between different XML formats, such as those arising out of the various tools, is made easier by the fact that XML formats are usually specified formally and by stylesheets in XSL (World Wide Web Consortium, n.d.–b), XML's standard language for declaratively specifying data transformations. Therefore, we concentrate on functionality that currently is not well supported and expect our software to be used in conjunction with other tools.

Ordinarily, when one thinks of software support, one thinks of end-user tools for coding data and analyzing the results. Although we could easily provide such tools for specific coding schemes—sets of codes and the permitted relationships among them—it is impossible to provide a uniform end-user interface for these functions that will always be usable. Different structures require different displays, data entry methods, and analytical techniques; in fact, when data sets are explored with many different kinds of codes, not one but several different displays and analyses may be appropriate. For this reason, we provide not end-user tools, but library functions that software developers can use to build tailored end-user tools efficiently. Our libraries, which are written in Java Version 1.4 (a common cross-platform programming language that works on Unix, Linux, Windows, and Macintosh OSX), include support for handling data, querying them (since this is what underpins analysis), and building interfaces. The toolkit also contains sample Java programs that demonstrate how to use the libraries, a simple and generic graphical user interface for running queries over a data set and returning the results, and an engine for building some limited interfaces from declarative specifications of what they should look like and how they should behave.

Data Handling in the NITE XML Toolkit

The NITE XML Toolkit provides library routines for loading and saving data in a data model that includes the sort of properties required to represent the example in Figure 1, accessing the loaded data, and changing it. Before we can describe this functionality in more detail, we must describe the data model that it implements.

The NXT data model. Our data model uses categories for describing a data set that designers will find natural. Data sets (or to use the linguistic terminology, *corpora*) consist 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* who take different roles in the interaction. 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). Our technology assumes that all signals for an observation are synchronized, so that the same time stamp refers to the same time on all of them, as is common with modern recording techniques.

We now turn to the coding of behaviors for an observation. Here, the basic unit is the *annotation*, which will have a type and may have any of four additional properties: textual content, attribute/value pairs further describing the annotation (where the attribute is a string and the value can be a string or a number), start and end times relating it to signal, and structural relationships to other annotations, linking them as parent and child or by some role, named using a string. The meaning of parenthood is that the children together make up the parent, with the natural implication for timing, whereas roles just convey some general connection. The type of an annotation can be represented either simply by a string or by using the special reserved role "type" to point to a *complex type*. As well as linking to other annotations, roles can be used to link to *objects* that represent something universal, such as the things in the room at which the agents can point or the lexical entries in a dictionary. Complex types and objects are treated very much like annotations in the data handling. Both can have attribute/value pairs and can link to annotations, complex types, or objects, using roles, but neither can have timing information. Complex types can have other complex types as children, which means that the child is a subclassification of the parent type; a resulting tree of complex types is called an *ontology*. Objects are contained in flat lists called *object sets*.

Annotations are organized into *layers* consisting of all of the annotations for a particular agent (or for the interaction as a whole) that either are of the same type or are drawn from a set of related types and that, 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 the annotations in them have children drawn from a single, "lower" layer, or *featural*, in which case annotations in them are related only to other annotations using roles but different roles can link into any layer they choose. Layers fit naturally into *codings*, or sequences of layers where each layer links into the next one in a hierarchical decomposition, ending either in a featural or a time-aligned layer. By "naturally," we mean that designers usually think of kinds of information to be added to a data set as tree structured in a way that can be described in terms of either a single coding or one coding per agent, all of which have the same layer structure.

In the data handling, intuitively, time-aligned and structural layers are temporally ordered, and timings percolate up from one or more different time-aligned layers through structural layers until they can go no higher. In the example data shown in Figure 1, for instance, the structural layers containing syntax annotations and the semantic frame inherit timings from the time-aligned word layer. The layer structure prohibits cycles and guarantees that this percolation will end. The children of an annotation in a structural layer are also ordered so that the first is structurally before the second, and so on. If the structural layer inherits its timings from some lower layer, the structural order of the children must coincide with the temporal one. Structural ordering is useful because not all annotations are timed. On the other hand, featural layers are unordered and merely gather behaviors into sets. A formal mathematical description of the model's structural and timing properties can be found in Evert et al. (2002).

The data model implementation (NXT Model). To use the NITE XML Toolkit's data handling, the data set designer must first write a "metadata" file, expressed in a particular XML file format, which formally describes the data set. This description is in the terms of the data model—that is, in terms of a number of observations, signals, agents, ontologies, object sets, codings decomposed into layers consisting of tags with particular types that have children in a particular layer, and so on. The metadata file also holds information about where on the computer to find the files containing the signals and the annotations, which are stored using a particular style of XML, called *stand-off annotation* (Carletta et al., in press), that employs one XML file per coding. Assuming this metadata has been authored, the implementation provides a library of Java routines for loading data, saving it, accessing what has been loaded, and making changes.

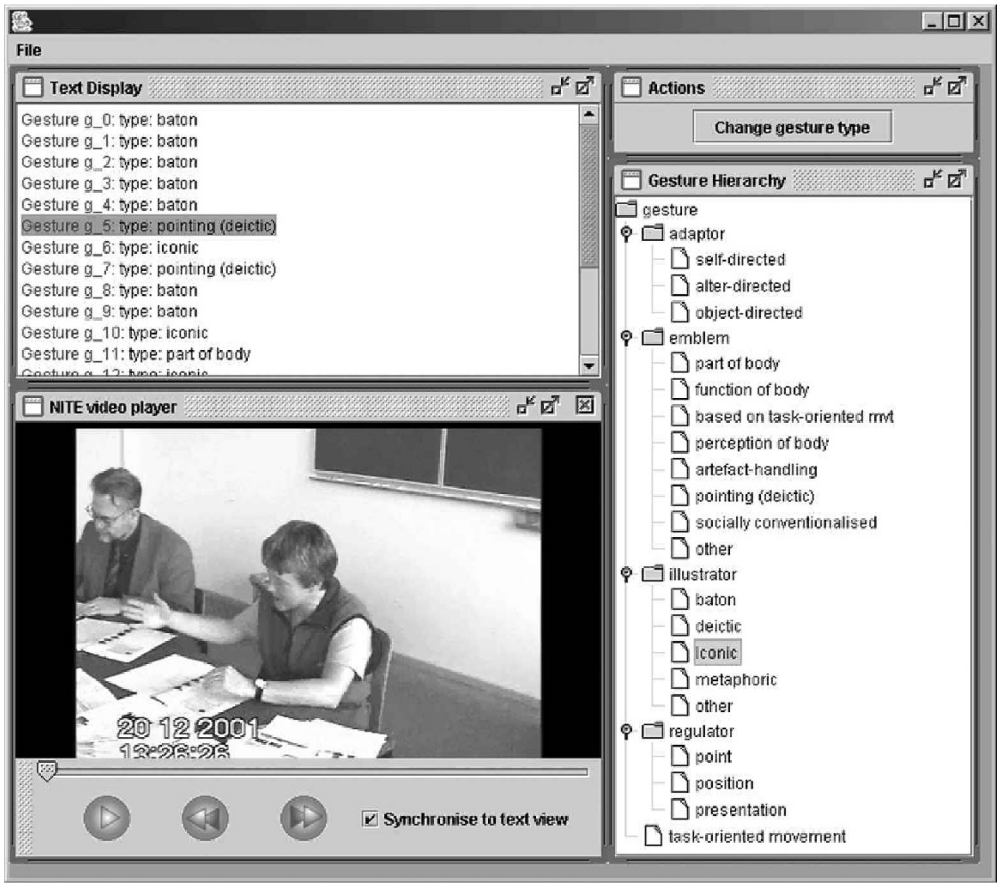
Loading can be done for all of the data in a data set, a single observation, or a particular set of codings. One can save either to the locations on disk from which the annotations were loaded or to a new one (in which case, a changed metadata file is also produced); there is also a choice about whether 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 annotations or to find an annotation's start and end times, parents, children, and roles. Typical routines for changing the data create new annotations (which are assigned to a particular layer, depending on their type and the agent whose behavior they describe), delete annotations, modify an annotation's attributes, and link two annotations by using the parent/child and role structures.

Data Querying

The next problem for working with this new sort of data pertains to analysis. 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 sequencing measurements. Again, the solution is to provide functionality that will help software developers support tailored analysis for end users. 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 fit our data model. This requires a specialist query language designed to exploit the relationships the model can express and an implementation that allows queries expressed in the language to be executed on loaded data.

The NXT query language (NQL). Queries in our query language, NQL, take the form of a number of variable declarations that either match all elements (annotations, complex types, and objects) or are constrained to match elements drawn from a designated set of simple types, followed by a set of conditions that are required to hold for a match, which are combined using the boolean operators NOT, OR, and AND. Queries return a structure that, for each match, lists the variables and the elements they matched. The essential properties of an element, beyond the simple type, are its identity, the attribute/value pairs, the timing, and the two types of structural links (parent/child and role). Accordingly, our query language has operators that allow match conditions to be expressed for each of these kinds of things.

The identity tests simply allow one to test whether two variables are bound to the same element or to different ones.



```
Gesture g_0: type: baton
Gesture g_1: type: baton
Gesture g_2: type: baton
Gesture g_3: type: baton
Gesture g_4: type: baton
Gesture g_5: type: pointing (deictic)
Gesture g_6: type: iconic
Gesture g_7: type: pointing (deictic)
Gesture g_8: type: baton
Gesture g_9: type: baton
Gesture g_10: type: iconic
Gesture g_11: type: part of body
```

Figure 2. A gesture-coding interface built using NXT.

The attribute tests include the ability to match all elements that have a particular attribute, regardless of value, or to match against the value of the attribute. Values can be tested against given strings or numbers or against the values found in other attributes, using equality, inequality, and the usual ordering tests (conventionally represented as <, <=, >=, and >). String values can also be tested against regular expressions, which 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 an element has timing information and whether the start or end time is <, <=, =, >=, or > a given time. The query language also has the following operators for comparing the timings of two elements for the same temporal extent: temporal inclusion, abutment, aligned (earlier or later starts or ends), and complete temporal precedence (where the end of the first element is before the start of the second one).

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 element is said to dominate another if it is possible to reach the second from the first by tracing zero or more child links. One element precedes another if they are ordered with respect to each other structurally (i.e., in terms of the ordering of children in related structural layers, not in terms of timing) and the first comes before the second.

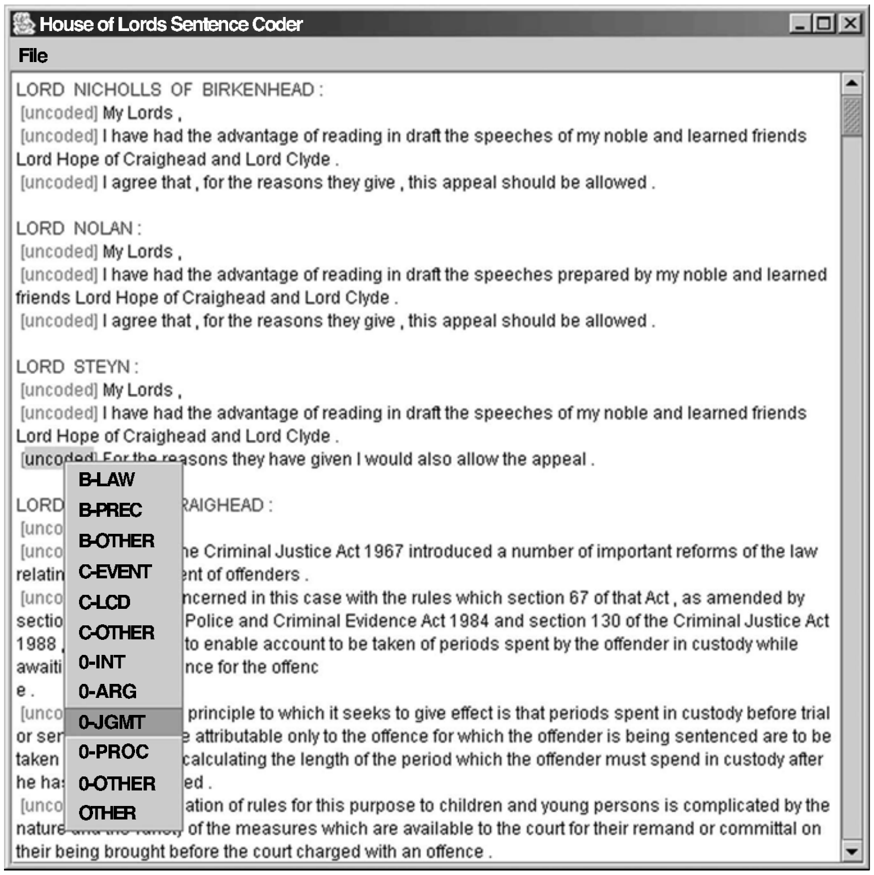
In addition, it is possible in NQL to constrain matches on the basis of quantified variables, using the standard quantifiers "exists" and "forall." Quantified variables are not returned in the query result.

Used in combination, these operators give rich and flexible access to a loaded data set. For example, one possible query for the data given in Figure 1 would be

```
($w word)($a): ($a >"antecedent" $w) &&
($w@pos =="PP$") && ($w@orth ~ /h.*/).
```

This query returns all pairs of words (bound with variable $w) and elements of any type (bound with the variable $a) where the word's part-of-speech attribute is PP$, the word's orthography attribute fits the regular expression /h.*/ (i.e., it has the letter "h" followed by zero or more other characters), and there is a pointer from the element to the word with the role antecedent. As another example,

```
($w word)(exists $s syntax)(exists $g gesture):
($s@cat=="PP") && ($s ^ $w) && ($w # $g)
```



LORD NICHOLLS OF BIRKENHEAD :
[uncoded] My Lords ,
[uncoded] I have had the advantage of reading in draft the speeches of my noble and learned friends Lord Hope of Craighead and Lord Clyde .
[uncoded] I agree that , for the reasons they give , this appeal should be allowed .

LORD NOLAN :
[uncoded] My Lords ,
[uncoded] I have had the advantage of reading in draft the speeches prepared by my noble and learned friends Lord Hope of Craighead and Lord Clyde .
[uncoded] I agree that , for the reasons they give , this appeal should be allowed .

LORD STEYN :
[uncoded] My Lords ,
[uncoded] I have had the advantage of reading in draft the speeches of my noble and learned friends Lord Hope of Craighead and Lord Clyde .
[uncoded] For the reasons they have given I would also allow the appeal .

LORD RAIGHEAD :
[unco
[unco he Criminal Justice Act 1967 introduced a number of important reforms of the law
relatin nt of offenders .
[unco ncerned in this case with the rules which section 67 of that Act , as amended by
sectio Police and Criminal Evidence Act 1984 and section 130 of the Criminal Justice Act
1988 , to enable account to be taken of periods spent by the offender in custody while
awaiti nce for the offenc
e .
[unco principle to which it seeks to give effect is that periods spent in custody before trial
or ser e attributable only to the offence for which the offender is being sentenced are to be
taken calculating the length of the period which the offender must spend in custody after
he ha ed .
[unco ation of rules for this purpose to children and young persons is complicated by the
nature of the measures which are available to the court for their remand or committal on
their being brought before the court charged with an offence .

Figure 3. A sentence classification interface built using NXT.

returns all words that are structurally dominated by prepositional phrases (i.e., by syntactic constituents with the category attribute PP) and also overlap temporally with any gesture. The query language is fully defined in Evert and Voormann (2002), and examples of its use can be found to accompany some of the example data sets that come with the software.

The query language implementation (NXT Search). Like our data model implementation, our query language implementation, called NXT Search, is in Java. The main function it contains allows a query to be evaluated on a loaded data set and can return either all matches or all the matches it finds up until some given maximum number is reached. The result is returned as a list structure amenable to further processing. The implementation also supports the conversion of what is returned into a simple XML format so that it can be saved easily.

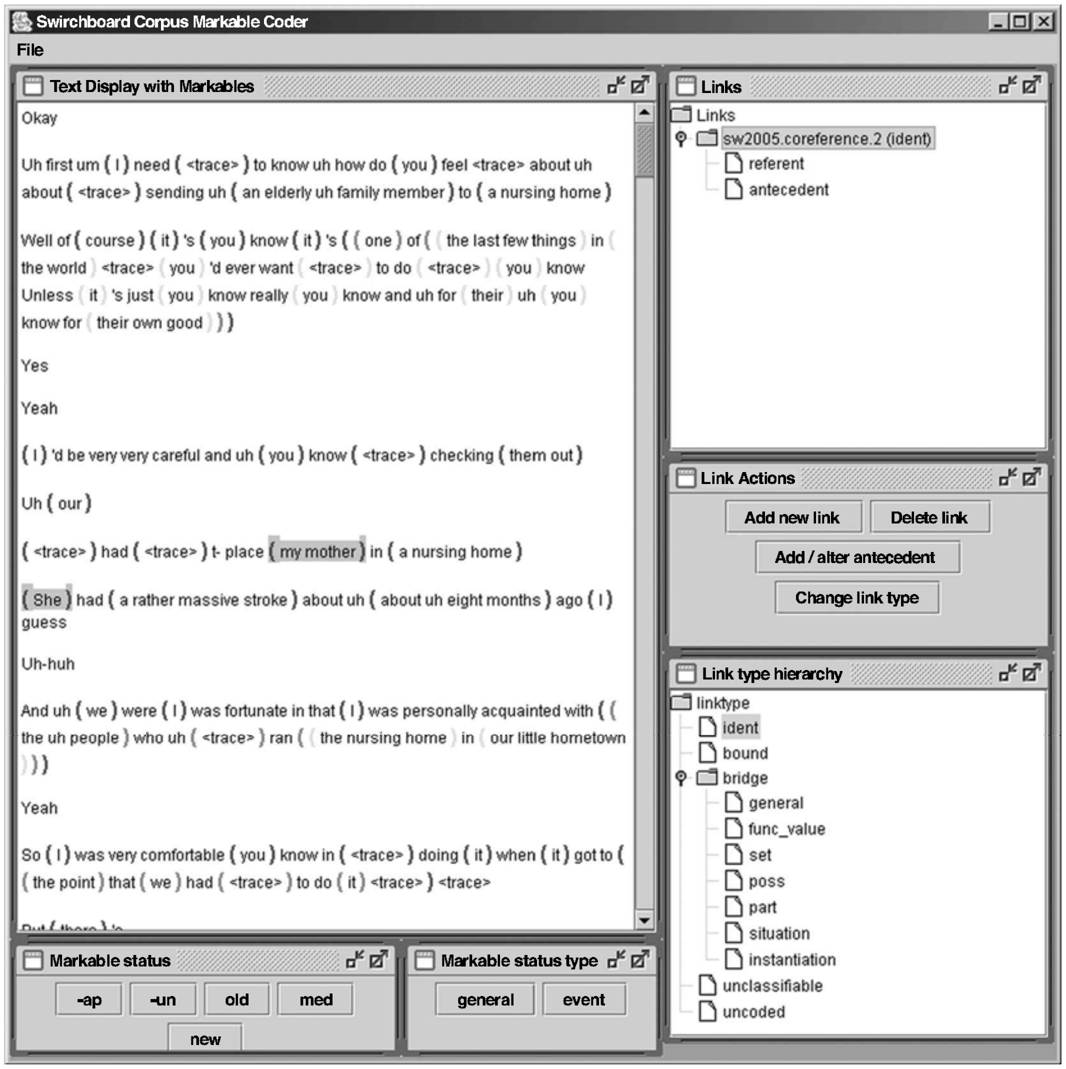
To make it easier to start using the toolkit, NXT Search includes a simple graphical user interface that will allow one to load a data set, type in a query, and display the results. The display simply lists the matches, allowing the user to hide or show the detailed variable bindings for each one, and tells the user how many there are. The results can be saved to an XML file. There are extensive help screens explaining the query language available from the interface menus, and when the query contains a syntax error, an error message pops up.

Together, our data-handling and data-querying functionalities provide the means of processing the new sort of data described in the Motivation: What Sort of Data? section. Our remaining software supports the building of graphical user interfaces for this data.

The Interface Library

Our interface library is closely allied to Java Swing (*Java Foundation Classes: Cross-Platform GUIs and Graphics*, n.d.), a common set of libraries for graphical user interfaces. Our library builds on top of Swing to construct the sort of tailored display components that are most useful for building data displays oriented around orthographic transcription. Our library does not contain components for building time-aligned displays of the sort that are familiar from observational analysis software, because many tools already support this style of display.

At the root of our displays is a resizable window. The main techniques we support for arranging this window are tabbed panes with different panels accessible using index tabs, split panes with two panels arranged vertically or



Okay

Uh first um { I } need { <trace> } to know uh how do { you } feel <trace> about uh about { <trace> } sending uh { an elderly uh family member } to { a nursing home }

Well of { course } { it } 's { you } know { it } 's { { one } of { { the last few things } in { the world } <trace> { you } 'd ever want { <trace> } to do { <trace> } { you } know Unless { it } 's just { you } know really { you } know and uh for { their } uh { you } know for { their own good } } }

Yes

Yeah

{ I } 'd be very very careful and uh { you } know { <trace> } checking { them out }

Uh { our }

{ <trace> } had { <trace> } t- place { my mother } in { a nursing home }

{ She } had { a rather massive stroke } about uh { about uh eight months } ago { I } guess

Uh-huh

And uh { we } were { I } was fortunate in that { I } was personally acquainted with { { the uh people } who uh { <trace> } ran { { the nursing home } in { our little hometown } } }

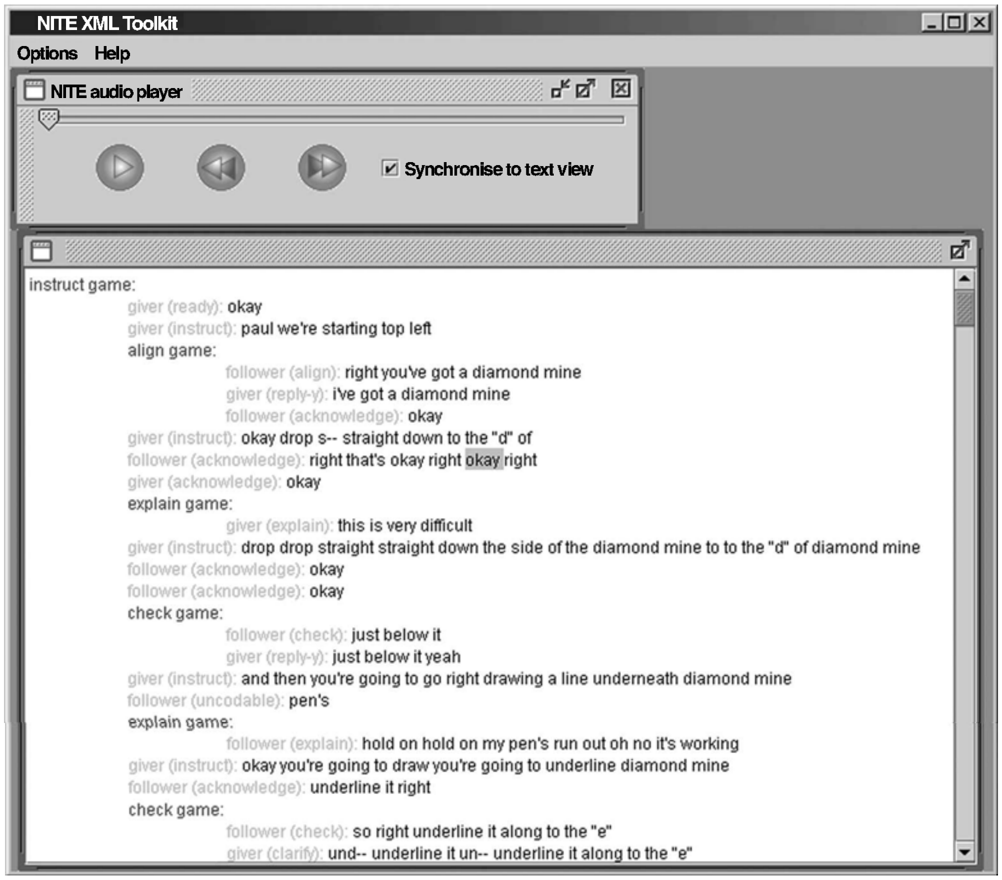
Yeah

So { I } was very comfortable { you } know in { <trace> } doing { it } when { it } got to { { the point } that { we } had { <trace> } to do { it } <trace> } <trace>

Figure 4. An information structure and co-reference coder built using NXT.

horizontally, scroll panes that automatically use scroll bars to access anything that cannot be displayed in the chosen size, or desktops, which in turn hold a number of independent resizable windows of these types. These display objects can be used to control the overall layout of the display. Within them, the library supports the use of vertical lists, trees with nodes that can be expanded or closed by the user (a technique familiar from Windows Explorer), grid panels that keep entries in a tabular layout, and text areas. Lists, grid panel entries, and the leaves of trees usually contain labels, which are simply bits of text presented in a particular manner. Labels can be configured in several ways, including specifying the font to use and the associated text to display when the label is moused over. Text areas, which are intended for larger texts, allow the user to define special font styles to use within them and contain text elements that, again, display bits of text and can be configured to use the defined styles. The labels and text elements are what display the actual data, as opposed to simply specifying how to lay it out. Although we have constructed them in order to build interfaces for data using our own data handling, they treat their text simply as strings and so can be used with data arising from other sources.

As well as the display objects, the library contains a signal player that can be used for audios and videos. This player, which has been built using Java Media Framework (*Java Media Framework API*, n.d.), has the usual basic controls: play, move to start, and so on. The player can also be synchronized with the other display windows. Some display objects (namely, a timed specialization of labels and normal text elements) can be given start and end times that specify how they relate to signal time. If the "synchronize to text view" button in a player is ticked, then as the signal plays, these objects will be highlighted when-



```
instruct game:
        giver (ready): okay
        giver (instruct): paul we're starting top left
        align game:
                follower (align): right you've got a diamond mine
                giver (reply-y): i've got a diamond mine
                follower (acknowledge): okay
        giver (instruct): okay drop s-- straight down to the "d" of
        follower (acknowledge): right that's okay right okay right
        giver (acknowledge): okay
        explain game:
                giver (explain): this is very difficult
        giver (instruct): drop drop straight straight down the side of the diamond mine to to the "d" of diamond mine
        follower (acknowledge): okay
        follower (acknowledge): okay
        check game:
                follower (check): just below it
                giver (reply-y): just below it yeah
        giver (instruct): and then you're going to go right drawing a line underneath diamond mine
        follower (uncodable): pen's
        explain game:
                follower (explain): hold on hold on my pen's run out oh no it's working
        giver (instruct): okay you're going to draw you're going to underline diamond mine
        follower (acknowledge): underline it right
        check game:
                follower (check): so right underline it along to the "e"
                giver (clarify): und-- underline it un-- underline it along to the "e"
```

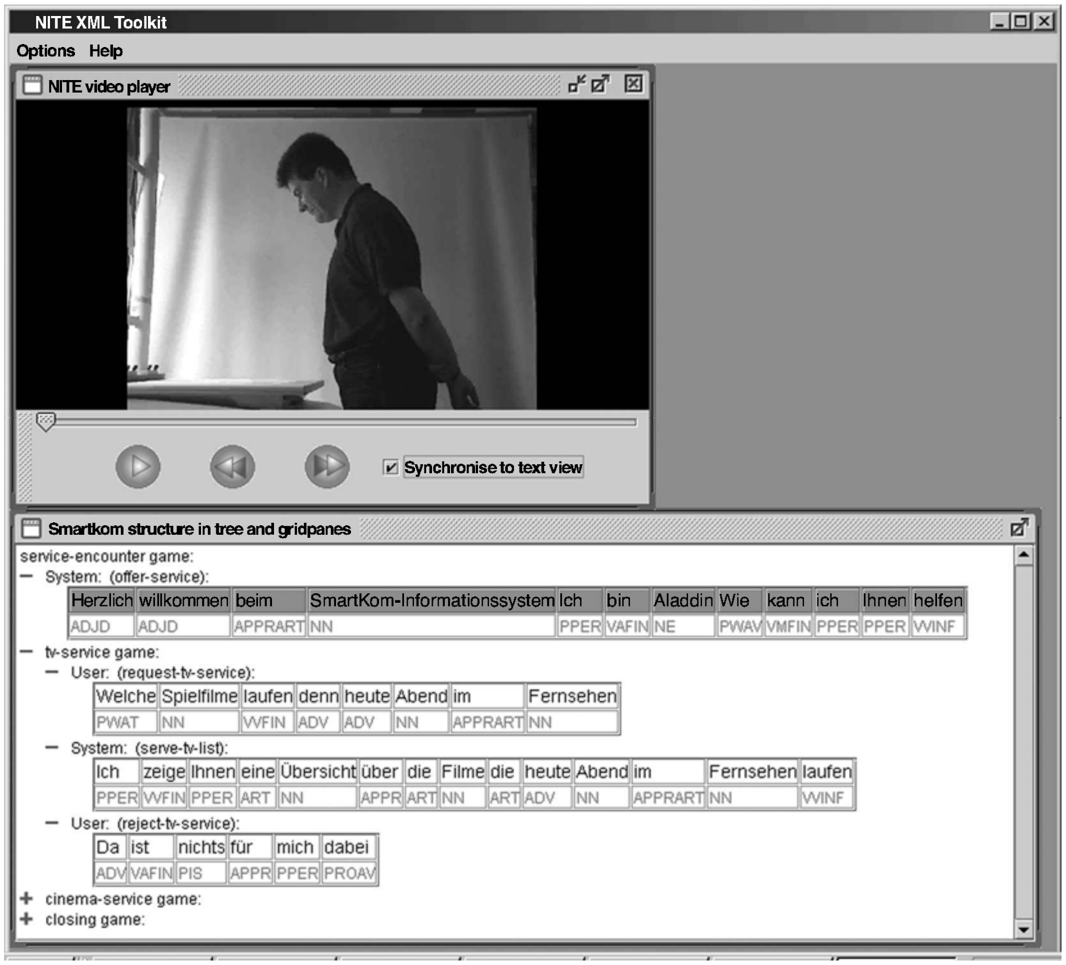
Figure 5. A dialogue structure display built using the interface engine.

ever the signal time is between their specified starts and ends. It is also possible to jump to the part of a signal that corresponds to a particular display object.

One way of handling user interaction that the library supports involves a standard interface device for each of a number of common changes to the data. Attaching the action to a display object enables the change for the data associated with that object. For instance, if a text element shows an attribute value, it is possible to attach an action to it that pops up a type-in window by which the user can change that value. Other typical standard actions are changing the textual content of an annotation, changing the value of an attribute by using a menu of the possible choices, and adding a new annotation as a child of the chosen one. Along with the standard actions, the programmer can drop through to Swing properties to control the interface behavior, with the usual array of buttons, menus, pop-ups, and other devices.

The Sample Programs

The NITE XML Toolkit supports software developers in the production of many kinds of tools for working with human interaction data sets: import and export into data formats for data exchange among tools, the addition of new annotations, a data display that includes construction of special purpose displays that emphasize specified properties of the data, and extraction of areas of the data that match specified properties. In order to make it easier for developers to see how to use the toolkit's libraries for this support, the toolkit includes a set of sample programs intended to demonstrate a range of end-user tools that match fairly common requirements. For instance, Figures 2–4 show screenshots of samples, built using technology, that come with the toolkit. Figure 2 shows an interface for coding gestures. The top left window highlights the current gesture as the video plays. At the time of the screenshot, the user has chosen a different gesture on the gesture ontology, displayed on the lower right-hand side, in preparation for changing the current gesture code. Figure 3 shows a simple interface for classifying sentence types in a textual document, as required for current text summarization research. Figure 4 shows an interface that, given a set of "markables" (primarily, noun phrases), allows the user to code them for their information status (new in the discourse, old, or mediated by a prior discourse entity) and the coreferential links between them. As well as interfaces, the samples include programs that process data—for instance, by running a query on the data model and adding a new layer of annotations based on the matches.



service-encounter game:
− System: (offer-service):

Herzlich	willkommen	beim	SmartKom-Informationssystem	Ich	bin	Aladdin	Wie	kann	ich	Ihnen	helfen
ADJD	ADJD	APPRART	NN	PPER	VAFIN	NE	PWAV	VMFIN	PPER	PPER	VVINF

− tv-service game:
− User: (request-tv-service):

Welche	Spielfilme	laufen	denn	heute	Abend	im	Fernsehen
PWAT	NN	VVFIN	ADV	ADV	NN	APPRART	NN

− System: (serve-tv-list):

Ich	zeige	Ihnen	eine	Übersicht	über	die	Filme	die	heute	Abend	im	Fernsehen	laufen
PPER	VVFIN	PPER	ART	NN	APPR	ART	NN	ART	ADV	NN	APPRART	NN	VVINF

− User: (reject-tv-service):

Da	ist	nichts	für	mich	dabei
ADV	VAFIN	PIS	APPR	PPER	PROAV

+ cinema-service game:
+ closing game:

Figure 6. An interface for displaying part-of-speech tagging, transcription, and dialogue structure, built using the interface engine.

The Interface Engine

The interface engine is a program that will create and run an interface for modifying the data set with a stylesheet, using an idea that was first prototyped in McKelvie et al. (2001). Starting at a given point in the input data, stylesheets contain a set of rules, with match conditions expressed in a query language, that specify what to put in the output and where in the input data to go next. The interface engine uses this mechanism to build a declarative specification of an interface that describes the display objects to use and what should go in them. This specification is, as usual, in XML. For instance, the declaration of a label containing the orthography of a word from Figure 1, when called from a rule that has matched on a word, might say

```
<Nite:DisplayObject type="Label" Fontsize="12">
 <xsl:value-of select = "@orth"/>
</Nite:DisplayObject>.
```

The engine then traverses this specification, building the interface as it goes.

Since stylesheets are relatively easy to write, the interface engine allows very different kinds of interfaces to be built quickly for the same data, tailored to specific uses. For instance, Figures 5 and 6 show displays built using the engine. Figure 5 shows dialogue structure coding, which relates the intentions behind individual utterances, with individual words synchronized to an audio signal. The main window uses a text area with indentations. Figure 6 uses a tree that contains grid panes and displays dialogue structure, transcription, and part-of-speech tagging simultaneously, where complete turns are synchronized with a video.

At present, the interface engine will work only on a single coding and does not use our own data-handling and -querying model but a simpler hierarchical data model (*JDOM*, n.d.) and the standard XPath query language (Clark & DeRose, 1999). These are completely standard technologies but do not allow for the inheritance of timing information, multiple parents of the same data item, or role relations. Since the interface engine is the most experimental part of our software provision, it is not yet always clear when using it is preferable to writing a Java program that uses the interface library directly. In our experience, stylesheets are much quicker to write than programs for the range of interfaces that the stylesheet language can describe and result in good end-user displays. However, especially for larger codings, the interactive behaviors for changing the data, which employ the standard actions defined in the interface library, can be too slow to use in practice. Therefore, at present, when the interface being written must change the data, the interface engine's main use is as a rapid prototyping device. It is technically feasible to overcome both the data-handling and the speed limitations, and therefore, we may be able to address these issues in future work.

Discussion

The NITE XML Toolkit provides support for data handling, querying, and the construction of user interfaces for data sets that contain both temporal and structural information. There are two other open-source projects that have aims similar to those of NXT—the Annotation Graph Toolkit (*AGTK: Annotation Graph Toolkit*, n.d.) and Atlas (*ATLAS Project*, 2000)[2]—but are perhaps less well developed.

The AGTK, which is oriented toward the needs of the speech community, defines a data model that is less expressive about structural relationships than NXT's but, accordingly, more appropriate for fast processing. It is written in C++, but there is a Java port available. 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 writing in C++ itself. The developers expect interfaces to call upon WaveSurfer (Sjölander & Beskow, n.d.), a compatible package, to display waveforms and play audio files.

Atlas is intended to be more generic than AGTK in allowing richer relationships between annotation and signal and permitting what are essentially arbitrary structures relating different annotations. The Atlas data model, which has been implemented in Java, is expressed at the level of individual data objects and how they relate to each other. Its developers are currently considering how best to support the definition of particular data sets in their model; once this has been done, it will be possible to compare what their modeling affords to what is available in NXT. The developers of Atlas plan both a query language and direct support for writing graphical user interfaces, and therefore, it is similar to NXT in ambition. However, these libraries are still being designed.

Once AGTK and Atlas are fully available, the choice of which of them to use will depend on the sort of data—in general, the simplest data model that fits naturally will be both fastest and easiest to understand—and the exact functionality implemented, as well as general criteria, such as how well the packages are documented and maintained. Meanwhile, NXT provides the widest overall support for working with multimodal data sets involving language. The common use of XML facilitates data interchange between tools built using NXT and external tools, including those that will be built using the other toolkits.

Since NXT is written in Java, we expect most users will write their applications in Java also. Although writing such applications will require Java and XML skills, these are increasingly common and usually are available wherever data sets are found. It is possible to call Java routines, such as those found in NXT, from other languages, although how one does so varies by platform and language. Because NXT is open source, it can be extended by the user community with new display objects, actions, and sample programs, as well as with interfaces for particular, widespread data sets, with users expected to share extensions, tools, and development tips.

The new tools that are becoming available provide new opportunities for multimodal data sets. Past work has tended to concentrate either just on language or just on what is observable from the video image, but it is becoming possible not only to do both, but also to integrate the two kinds of information in a meaningful way. As the developments progress, language researchers are likely to come to understand the utility of considering all the modalities by which people communicate, whereas observational analysts are likely to begin deeper explorations of the content of an interaction. Thus, these tools facilitate a step-change in current research methods.

REFERENCES

AGTK: Annotation Graph Toolkit (n.d.). Retrieved May 26, 2003 from http://agtk.sourceforge.net/.

ATLAS Project (2000, Rev. February 6, 2003). Retrieved May 26, 2003 from http://www.nist.gov/speech/atlas/.

ATLAS.ti (n.d., Rev. February 18, 2003). Retrieved May 26, 2003 from http://www.atlasti.de/.

Bales, R. F. (1951). *Interaction Process Analysis: A method for the study of small groups.* Cambridge, MA: Addison-Wesley.

Barras, C., Geoffrois, E., Wu, Z., & Liberman, M. (2001). Transcriber: Development and use of a tool for assisting speech corpora production. *Speech Communication*, **33**, 5-22.

Carletta, J. C., McKelvie, D., Isard, A., Mengel, A., Klein, M., & Møller, M. B. (in press). 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: Continuum International.

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

Day, D., Aberdeen, J., Hirschman, L., Kozierok, R., Robinson, P., & Vilain, M. (1997). Mixed-initiative development of language processing systems. In *Fifth Conference on Applied Natural Language Processing*. Washington, DC: Association for Computational Linguistics.

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

Evert, S., & Voormann, H. (2002). *NXT Query Language.* Retrieved May 26, 2003 from http://www.ltg.ed.ac.uk/NITE/.

Goodwin, C. (1981). *Conversational organization: Interaction between speakers and hearers.* New York: Academic Press.

Java foundation classes: Cross-platform GUIs and graphics (n.d., Revised April 12, 2003). Retrieved May 26, 2003 from http://java.sun.com/products/jfc/.

Java Media Framework API (n.d., Revised May 6, 2003). Retrieved May 26, 2003, from http://java.sun.com/products/java-media/jmf/.

JDOM (n.d.). Retrieved May 26, 2003 from http://www.jdom.org/.

Kipp, M. (2001, September). *Anvil: A generic annotation tool for multimodal dialogue*. Paper presented at the Seventh European Conference on Speech Communication and Technology (EUROSPEECH), Aalborg.

Language Technology Group (n.d., Revised March 30, 1998). *LTG software: LT-POS*. Retrieved May 26, 2003 from http://www.ltg.ed.ac.uk/software/pos/index.html.

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

Milde, J.-T., & Gut, U. (2001). The TASX-environment: An XML-based corpus database for time aligned language data. In S. Bird, P. Buneman, & M. Liberman (Eds.), *Proceedings of the IRCS Workshop on Linguistic Databases* (pp. 174-180). Philadelphia: University of Pennsylvania.

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.

Silverman, K., Beckman, M., Pitrelli, J., Ostendorf, M., Wightman, C., Price, P., Pierrehumbert, J., & Hirschberg, J. (1992). TOBI: A standard for labeling English prosody. In *International Conference on Speech and Language Processing (ICSLP)* (Vol. 2, pp. 867-870) Alberta: Permanent Council for the Organization of International Conferences on Spoken Language Processing.

Sjölander, K., & Beskow, J. (n.d., Revised May 9, 2003). *Wavesurfer*. Retrieved May 26, 2003 from http://www.speech.kth.se/wavesurfer/.

World Wide Web Consortium (n.d.–a). *Extensible Markup Language (XML)*. Retrieved May 26, 2003 from http://www.w3.org/XML/.

World Wide Web Consortium (n.d.–b, Revised January 16, 2003). *The Extensible Stylesheet Language (XSL)*. Retrieved May 26, 2003 from http://www.w3.org/Style/XSL/.

NOTES

1. For instance, The Observer is expected to ship with XML export functionality in its next major release (L. P. J. J. Noldus, personal communication, April 1, 2003).

2. This Atlas is not the same as ATLAS.ti (n.d.), an end-user tool popular for textual annotation that has recently incorporated some video functionality.

(Manuscript received November 5, 2002;
revision accepted for publication May 18, 2003.)