# The MATE Workbench - an annotation tool for XML coded speech corpora

David McKelvie [a], Amy Isard [a], Andreas Mengel [b],
Morten Baun Møller [c], Michael Grosse [c] and Marion Klein [d]

[a] *Language Technology Group, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9LW. email dmck@cogsci.ed.ac.uk*

[b] *Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart*

[c] *Natural Interactive Systems Laboratory, Odense University*

[d] *Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken*

**Abstract**

This paper describes the design and implementation of the MATE workbench, a program which provides support for the annotation of speech and text. It provides facilities for flexible display and editing of such annotations, and complex querying of a resulting corpus. The workbench offers a more flexible approach than most existing annotation tools, which were often designed with a specific annotation scheme in mind. Any annotation scheme can be used with the MATE workbench, provided it is coded using XML markup (linked to the speech signal, if available, using certain conventions). The workbench uses a transformation language to define specialised editors optimised for particular annotation tasks, with suitable display formats and allowable editing operations tailored to the task. The workbench is written in Java, which means that it is platform-independent. This paper outlines the design of the workbench software and compares it with other annotation programs.

**Zusammenfassung**

Dieser Beitrag beschreibt das Design und die Implementierung der MATE Workbench, einem Programm für die Annotation von gesprochener und geschriebener Sprache. Die Workbench bietet Funktionen für die flexible Darstellung und Bearbeitung von Annotationen sowie komplexe Suchanfragen an vorhandene Korpora. Die Workbench hat einen offeneren Ansatz als die meisten existierenden Annotations-Werkzeuge, welche oft auf ein festes Annotationsschema zugeschnitten sind. In der MATE Workbench kann jedes Annotationsschema benutzt werden, so es in XML kodiert und - wenn verfügbar - mit einem Sprachsignal verbunden ist. Die Workbench nutzt eine Transformationssprache für die Definition und Generierung des für die jeweilige Aufgabe angemessenen Editors mit entsprechenden Anzeigeformaten und Bearbeitungsfunktionen. Die Workbench ist in der Programmiersprache Java geschrieben und somit platform-unabhängig. Der Beitrag erläutert das Design der Workbench-Software und vergleicht es mit dem anderer Annotationsprogramme.

*Preprint submitted to Elsevier Preprint* *2 May 2000*

**Résumé**

Nous présentons, dans cet article, la conception et le développement du workbench MATE, un programme pour l'annotation de la parole et des textes écrits. Il facilite la visualisation et l'édition flexibles de telles annotations, et permet des requêtes complexes sur un corpus résultant. Le workbench offre une approche plus souple que la plupart des outils d'annotation, qui ont souvent été conçus pour un jeu d'étiquettes spécifique. Le workbench MATE permet le traitement de tout jeu d'étiquettes pourvu que ce dernier puisse s'exprimer en format XML (rattaché au signal vocal, si disponible, en utilisant certaines conventions). Le workbench utilise un langage de transformation pour définir les éditeurs spécialisés qui sont optimalisés pour des tâches d'annotation particulières avec des visualisations appropriées et des actions permises conçues en fonction de la tâche. Le workbench est écrit dans le langage Java, ce qui le permet d'être indépendant de la plateforme. Cet article décrit l'architecture du logiciel du workbench et le compare à d'autres outils d'annotation.

## 1 Introduction

The MATE workbench is designed to aid the annotation of linguistic corpora and to help users explore the relationships among different structures within such a corpus. The task we wish to support is the annotation of spoken dialogues, which means we need to support structured annotations on text as well as relating these to the speech signal where available. Annotation is a tedious job for humans. Although much research in computational linguistics has been devoted to developing trainable automatic algorithms for annotating corpora, it still remains the case that such algorithms often require hand annotated training data and human post-checking to ensure accuracy. Using an annotation tool which is specialised to the annotation task at hand, i.e. one which only shows the information relevant to the task, and only allows those editing actions which contribute to the task (and optimises the human actions required), is probably preferable to a general purpose editor. However, this normally requires writing individual editors for each task.

The MATE workbench provides a general framework for defining specialised annotation editors, and makes the writing of an editor tool for a particular annotation scheme and particular annotation task relatively easy. This generality is provided by allowing the use of any annotation schema coded in XML [Bray et al., 1998] and by allowing the corpus designer to write rule based transformations (using the XSLT language [Clark, 1999]) which describe how the corpus is presented to the annotator and what editing actions are available. The workbench provides a number of pre-defined stylesheets for use with particular annotation schemes, but its major strength is that it is possible to write new stylesheets for existing or new schemes. We believe that the stylesheet language is sufficiently high-level for writing stylesheets to be significantly easier than writing an editor from scratch. Using the workbench, the user can display and edit existing XML-encoded corpora, add new levels of annotation, perform queries over part or all of a corpus, and display or output the results.

In this paper we first present a review of a number of other annotation tools, followed by an overview of the approach we have taken in the MATE workbench. This is followed by descriptions of each of the major system components and the user interface. We then present examples of the workbench in action, and finish with a discussion and possible directions for future work.

## 2 Review

There are many existing tools which offer support for annotation, querying and/or display of speech corpora, and an extensive list of these, with links, can be found at the Linguistic Data Consortium's Linguistic Annotation web site [Bird and Liberman, 2000b]. These tools have been created for many different and sometimes very specific purposes, and the MATE workbench does not aim to replace them all, but to offer a way of creating specific interfaces more easily in future, and to provide a framework which makes it possible to work with many different annotation schemes which are written in a common format (i.e. XML).

Firstly, there are a number of tools which provide detailed interfaces for transcription and manipulation of the speech signal. The **Emu** system [Cassidy and Harrington, 2000] is written in C++ and runs under Windows, Solaris and Linux, and accepts many sound formats if run under Unix with the Edinburgh Speech Tools Library [Taylor et al., 1999]. The user can display a speech signal in various different formats, and annotate it with labelled segments or events, and it is also possible to view and edit the labels displayed in a hierarchical tree-like structure. The annotations are saved in the same format used by Entropic Xlabel (see below) and a script has been written to perform conversion to and from the BAS Partitur format [Schiel et al., 1998]. Emu also includes a query tool which allows the user to save annotations in a format which can then be used for SQL database queries. **Praat** [Boersma and Weenik, 2000] runs under all common operating systems, contains many speech analysis algorithms, and also allows labelling of time intervals and points on multiple tiers. It accepts many sound formats, and contains a large library of speech analysis tools. Annotations are saved in a format specific to this tool. **Transcriber** [Barras et al., 2000] is written in Tcl/Tk and runs under Unix, Mac and Windows. Using the Snack [Sjölander, 2000] sound extension, it accepts most common sound file formats, and can process very large audio files. The speech signal can be displayed in various ways, and segmentation and labelling windows are synchronised with the speech display. Annotation is possible according to some pre-defined annotation schemes; annotations are saved in XML format. It is possible to transcribe overlapping speech, but the other annotation levels are strictly hierarchical. **Entropic waves+** and **Xlabel** [Entropic, 1996] have frequently been used in the past for annotation of speech signals, but future support is uncertain at present.

Secondly, there are a number of systems which focus on higher level annotation. The **Alembic** workbench [Day et al., 1997] is designed to facilitate the construction of training corpora for information extraction. Different annotation schemes can be used with the workbench, but only one annotation can be displayed at a time, and there is a fixed annotation environment and view of annotations. It uses a Tipster compliant format [Grisham, 2000], and can also output annotations in an SGML format, but it does not provide facilities for listening to or manipulating speech signals. The **LACITO Archivage tool** [Jacobson et al., 2000] displays recordings and annotations made by LACITO members over the last thirty years; these annotations are stored as XML, and XSL stylesheets are used to provide multiple views of the same data. Displays are constructed from Java applets, and can be displayed in any standard web browser.

Thirdly, there are systems which aim to bring together existing tools. **GATE** [Cunningham et al., 1996] supports modules written in any programming language. It uses the Tipster architecture internally, but also supports SGML input and output, and it is now possible to do manual annotations as well, with a single fixed interface. GATE also contains a tool for comparing annotations, for example one done manually and one automatically on the same text. **EUDICO** [Grugman and Russel, 1999] is platform independent, written in Java, and can be used over the internet with any standard browser. It allows the user to synchronise text presentation with the playback of MPEG video. It is designed so that new tools and corpora can easily be added. It is currently available only in “read-only” version (i.e. no annotation is possible). A successful pilot study was carried out to investigate the possibility of a merger of EUDICO with components from GATE.

Three of the tools described above (Emu, Praat and Transcriber) provide speech signal manipulation tools far more sophisticated than those currently available in the MATE workbench. We envisage that for some purposes, researchers might wish to do the initial transcription from the speech signal using another tool, and then use the MATE workbench for subsequent levels of annotation, and for creating displays using these annotations. Except in the case of Transcriber, this would necessitate transforming the annotations into XML. Note that a tool for converting Entropic Xlabel format to XML is already provided with the MATE workbench (see section 6). Alembic and

LACITO both provide similar functionality to parts of MATE, but tailored to the needs of their particular projects. Alembic provides automatic annotation options far beyond the scope of MATE. LACITO takes the most similar approach to MATE of all the tools studied, but it is restricted to one particular domain; we would hope that in future if a project has need of such a tool, they could more easily build it by writing MATE stylesheets than by building an entire tool from scratch. GATE and EUDICO aim to bring together existing tools and make it easier to use them in a common framework. The MATE workbench also intends to facilitate the display of output of automatic analysis tools - it would be useful to investigate possibilities of integrating the MATE workbench with one or both of these tools. The workbench has much in common with other XML editors and browsers currently being developed, such as the Amaya HTML browser/editor [Vatton et al., 1999]. We go beyond the scope of Amaya by supporting arbitrary annotation schemes rather than just HTML and by using a more powerful mapping between document and display structures.

## 3 Architecture

In this section, we first describe the way that XML is used to represent annotations, and then the format in which the XML data are stored internally in the MATE workbench.

### *3.1 XML*

XML has proved to be a widely used and effective format for annotating text and speech corpora, and a number of projects, including the Text Encoding Initiative (TEI) [Ide and Véronis, 1995] and the Corpus Encoding Standard (CES) [Ide et al., 2000], have developed general purpose SGML standards for linguistic annotation. XML's flexible hierarchical structure matches well with many kinds of linguistic representation, and the MATE workbench uses XML as its input/output format, and uses a similarly structured internal data model. However, the strictly hierarchical nature of XML is at odds with certain aspects of linguistic (particularly speech) data; in multi-speaker dialogues, speech may overlap, and different annotation hierarchies coded on a corpus may also overlap, such as prosody and syntax.

One way to indicate this non-hierarchical structure in XML is by the use of standoff annotation [Isard et al., 1998]. Linking between elements is done by means of a distinguished *href* attribute of elements, which uses a subset of the XPOINTER and XLINK proposals [DeRose et al., 1999,DeRose et al., 2000]

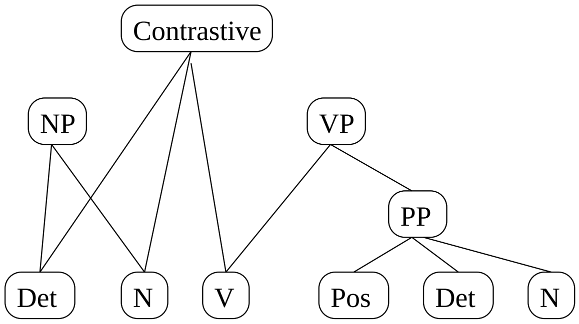


Fig. 1. Internal representation of example with intersecting hierarchies

to point to arbitrary elements in the same or different files. Such attributes are often called hyperlinks. The TEI and CES schemes propose a number of ways of representing intersecting hierarchies and overlapping speech segments, which usually use XML attributes of type ID/IDREF to build non-hierarchical links between elements in the same document. It would be possible to use such a mechanism in the workbench, although we have preferred to use hyperlinks (which are ID/IDREF links extended to deal with references in different files and to ranges of elements) and to use explicit start and end attributes to represent time.

Our extended data model allows us to represent overlapping or crossing annotations. The following quasi-XML (which is not well-formed according to the XML specification) would represent a case where a contrastive marking is on the subject and verb and crosses a <vp> constituent.

```
<np><contrastive><det>the</det><n>cat</n></np>
<vp><v>sat</v></contrastive> <pp><pos>on</pos>
<det>the</det><n>mat</n></pp></vp>
```

This example could be represented in well-formed XML as shown below.

```
<np><det id="x1">the</det><n id="x2">cat</n></np>
<vp><v id="x3">sat</v><pp><pos id="x4">on</pos>
<det id="x5">the</det><n id="x6">mat</n></pp></vp>
<contrastive href="id(x1)..id(x3)"/>
```

When the above XML is read into the MATE internal representation (see section 3.3), a normal tree structure is built for the <np> and the <vp> elements. When the <contrastive> element is read and a node for it is created, because its *href* attribute refers to already existing elements, it is linked to the corresponding already existing nodes. The workbench, in effect, interprets the input XML to convert *href* attributes into references to other elements, providing a way to build non-tree structures (see the discussion in section 3.3). The final internal representation of this example is shown in figure 1. Note that the <contrastive> element could be in a different file from the other elements.

Standoff annotation (where hyperlinks can refer to elements in the same or a

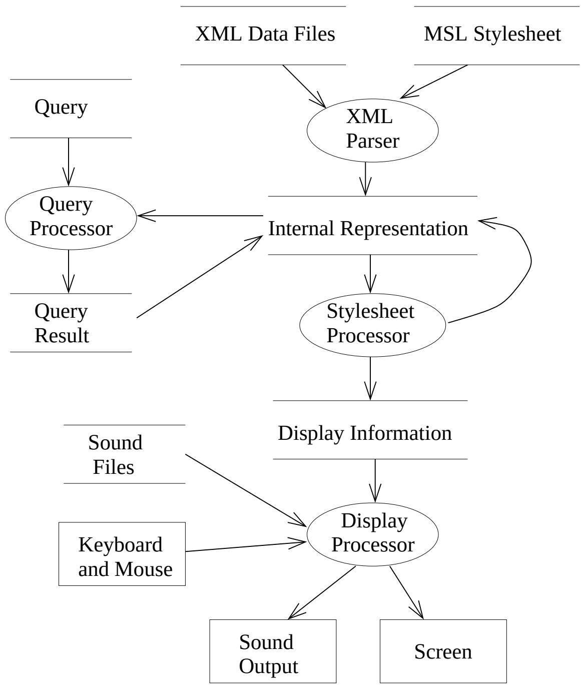


Fig. 2. MATE workbench software architecture

different file) is not essential, but it provides a way of factoring annotations and is very useful if multiple versions of an annotation layer are required, for example to compare annotations by different coders.

### *3.2 Software Architecture*

The architecture of the MATE workbench consists of the following major components, as illustrated in figure 2:

**An internal database** which is an in-memory representation of a set of hyperlinked XML documents. There are functions for loading and outputting XML files into and out of the database (see section 3.3).
**A query language and processor** which are used to select parts of this database for subsequent display or processing (see section 4).
**A stylesheet language and processor** which respectively define and implement a language for describing structural transformations on the database. The output of a transformation applied to a document can either be another document in the database or a set of display objects. These are used to define how the database will be presented to the user (see section 5).
**A display processor** which is responsible for handling the display and editing actions. This takes the display object output of a stylesheet transformation and shows it to the user (see section 5.3).
**A user interface** which handles file manipulation and tool invocation (see

section 6).

The workbench is written entirely in Java 1.2, which means that it is platform-independent. We have tested it on Unix (Solaris) and Windows (NT and 98) but it does not currently run on Macintosh computers because there is not yet a version of JDK1.2 for this platform.

There is always a tradeoff between generality and specificity in software design, and specialised facilities in annotation tools (e.g. automatic annotation) will always require the writing of specialised software. However, we think that it is worthwhile to have a common framework that supports file I/O, database handling, query support and flexible display facilities into which this software can be slotted, rather than building specific tools. No one tool can do everything, so the MATE workbench allows the reuse of other software, either by using XML as a data interchange format, or by calling other Java modules which access the internal representation and the MATE display structure using defined interfaces.

### *3.3 The Internal Representation*

At the heart of the workbench is a database which contains a representation of the XML files which have been loaded into the workbench. The abstract data model that we use is a directed graph, and any directed graph can be represented in the model. Each node in this graph has a type name (e.g. *word*, *phrase*, or other arbitrarily chosen name), a set of string-valued attributes and a list of *child* nodes (corresponding to outgoing edges from the node). Edges in the model are unlabelled. A node may have several different *parent* nodes, but each node has (at most) one distinguished *parent* node which can be used to treat the graph as an edge-disjoint set of trees. This view of the graph is useful for algorithms which require a tree structure, such as writing the internal representation out to a set of files, applying the stylesheet processor, or asking questions about the linear order of two elements.

The semantics or interpretation of the data model is flexible and may vary depending on the requirements of the annotation schemes used. A common, but not necessary, interpretation of a node and the parent-child relation is that a node represents a segment of speech and its children represent a finer sub-division of the same segment of speech, where the order of the children corresponds to the temporal order. Links to speech or other data files are stored as attributes of these nodes (for example as time offsets). But there is no reason why one could not have an element <or> whose children represented alternative descriptions of the same segment of speech.

The mapping from XML-encoded files to this data model is fairly straightforward. As each XML file is read into the database, each XML element and piece of text is represented as a new node, whose type is the name of the XML element (or "#PCDATA" for text). Attributes are copied over (taking default values from the XML Document Type Definition (DTD) if required). The parent-child relation is that derived from textual inclusion in the XML files, and the distinguished parent of a node is the node of the textually enclosing XML element. This gives us a set of trees, one per loaded file. As a final stage of loading, elements which are hyperlink sources (i.e. which have *href* attributes), have additional (non-distinguished) parent-child links added to them. The *href* attribute is evaluated to give a list of other nodes in the internal representation, and each node in this list is then added as a new child of the source node. This gives us the ability to represent arbitrary directed graphs in XML. The expansion and evaluation of hyperlinks is not strictly necessary, but it makes for a more uniform representation, as textual inclusion of elements and hyperlinks are treated on an equal footing. This makes it easier to evaluate queries which traverse hyperlinks.

Outputting the contents of the internal representation back to a set of XML files consists of writing a file for each root node (those with no distinguished parents) and writing each descendant out recursively as XML, replacing non-distinguished children by hyperlinks. Normally these hyperlink *href* attributes can be added to existing elements but in some cases, new dummy elements must be created to hold the attribute. For example, if an element has three children, the first and third of which are in the same document as the parent, but the second of which is in a different file, then we need a dummy element between the first and third elements which references the element in the different file. This does not change the contents of the internal representation once these files are read back in, as the loading software can recognise and replace the dummy elements by a reference to the correct element .

### *3.4 Implementation of the Internal Representation*

The internal representation is implemented as a database of triples of <*node identifier, property name, property value*>. Properties generalise the attributes of an XML element and most are string valued, but some (e.g. the **children* and **anyparent* properties) have values which are lists of references to other nodes in the internal representation. As an extension to the standard Document Object Model (DOM) [Wood et al., 2000], DTDs are also represented as objects in the database. The types of nodes and their interrelationships are shown in figure 3. This representation of the database makes it easy for the query language (see section 4) to express general questions about the annotations.

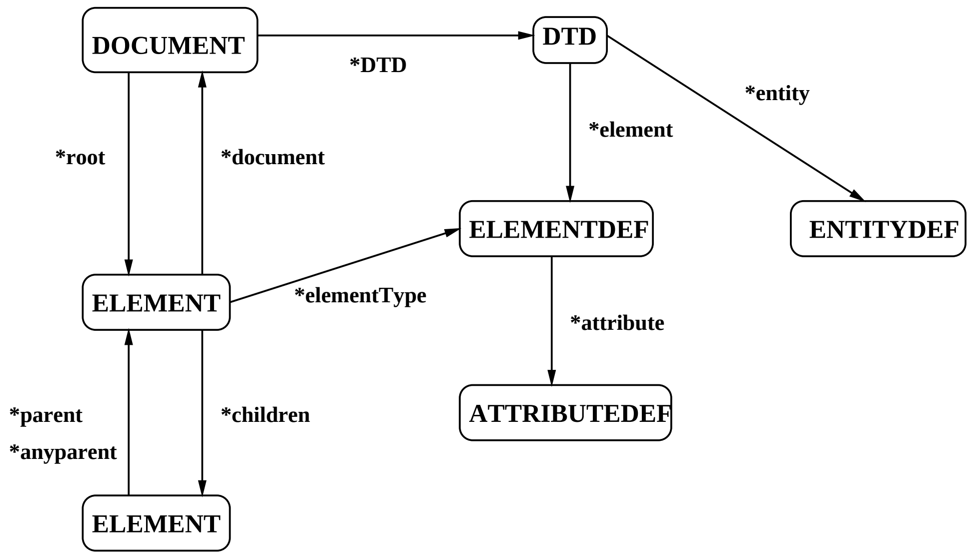


Fig. 3. A partial schema for the MATE internal representation

Most uses of this data model for describing corpora so far have resulted in graphs which are acyclic (i.e. ones in which by following child pointers from a node one never reaches the node or any of its ancestors again), but we are not yet sure whether we want to enforce this restriction. Our data model is similar to the semistructured data model proposed by the Lore project [Goldman et al., 1999], which has also been used as a way of representing XML. However we use XPOINTER [DeRose et al., 1999] attributes rather than IDs for hyperlinking, since that allows us to link across files without assuming a single ID namespace across all files. In addition we amalgamate elements linked to by *href*s and textual children into the same parent-child relation.

A set of XML files is a valid MATE annotation if each file is an XML document which is valid with respect to its DTD. We do not currently check any restrictions on hyperlinks between the files, and there is therefore no way to ensure that all hyperlinks from utterance annotations reference word annotations. It would be relatively easy to add this further level of checking to the workbench, to make sure that the children of each node are locally well-formed after hyperlinking. However, in the present version, one could write a stylesheet that uses the query language to find words that violate this constraint (i.e. an utterance that contains a non word) and display them in an error window. This approach to extended XML validity checking is similar in spirit to the Schematron proposal [Jelliffe, 2000].

An important point in the workbench design is that it is reflexive; all information about the system is kept in a similar format in the internal representation. Query results and stylesheets, which describe the appearance of the user interface are all stored in the same format. This approach has a number of advantages. One of these is that we could use the workbench itself to provide an editor for writing stylesheets, or transformations could be used to convert stylesheets into other stylesheets in a programmatic way. More importantly, since the internal representation structure is isomorphic to the structure of an XML file, query results can be output as XML files, and can be displayed to the user in a number of formats according to the stylesheet used. In particular, since query results contain pointers to the elements that satisfy the query, we have the option of either displaying results separately from the corpus or of highlighting elements in context.

## 4 The MATE Query Language Q4M

The linguistic annotation of speech data is not an end in itself, but an investment for later inspection and analysis of the data. The purpose of performing a query on this annotation may not only be the identification of explicitly encoded information, but also the retrieval of information that can only be derived by manipulation of existing information [Heid and Mengel, 1999]. This implies that we need a query language that is both suited to our data model and sufficiently expressive to allow a wide range of queries.

XML is now being used for purposes and domains previously covered by relational databases [Abiteboul et al., 2000]. Because XML is strongly hierarchical and its structure is flexible (it is possible, for example, to have optional or repeated elements), its structure is not quite in line with relational models which prefer a fixed schema and a normalised form. This has led to a large number of proposed XML query languages [Marchiori, 1998] more suited to the data model. Similar work has also been done in the database community on developing query languages for data models which are more flexible than relational models, such as the work on semistructured databases by the Lore project [Goldman et al., 1999].

We nonetheless chose to develop our own query language and processor (Q4M). The reasons were: (1) when we started our work, there was no XML query module available in Java; (2) there is still no standard in this area; (3) we needed the query processor to interact well with our internal database; and (4) we needed an extended data model that was an arbitrary directed graph and not just a tree structure. [Murata and Robie, 1998,Abiteboul et al., 2000] contain discussions of general considerations of query languages in the XML world. A query language appropriate for the data model must make XML constructs like element, attribute and attribute value accessible. We need to be able to access elements either directly, via some constraint, or indirectly by following the parent-child links in the database. In addition, the order of children in the data model is linguistically significant, so we need to query on

| | | |
|---|---|---|
| (1): | `($p pros)` | `$p` refers to <`pros`> elements |
| (2): | `($s sent)` | `$s` refers to <`sent`> elements |
| (3): | `($w word);` | `$w` refers to <`word`> elements |
| (4): | `($s.type ~ "ans") &&` | <`sent`> `type` attribute value is "ans" AND |
| (5): | `($s.who ~ "Mary") &&` | <`sent`> `who` attribute value is "Mary" AND |
| (6): | `($s ][ $w) &&` | <`sent`> immediately precedes <word> AND |
| (7): | `($w.pos ~ "adv" ) &&` | <`word`> `pos` value is "adv" AND |
| (8): | `($w.who ~ "Peter") &&` | <`word`> `who` value is "Peter" AND |
| (9): | `($w @ $p) &&` | <`pros`> element occurs during the <`word`> AND |
| (10): | `($p.type ~ "H*")` | <`pros`> `type` value is "H*" |

Fig. 4. An example Q4M query.

this order.

We also want to be able to return tuples of elements, that is combinations of elements with specific properties, pairs of elements with comparable properties, elements in a hierarchical relation and so on. Some query languages such as XML-QL [Deutsch et al., 1998] and SgmlQL [LeMaitre et al., 1996] do provide result tuples but this feature goes beyond some other XML query languages (such as the one defined in XSLT) which are restricted to returning lists of elements. The following example is a query which requires tuples:

> *Find all adverbs spoken by Peter which include an "H*" accent, and follow directly after an answer by Mary.*

Figure 4 shows the equivalent query expression in Q4M syntax. The Q4M expression has a variable definition part (1-3) and a query constraint part (4-10). Various mathematical operators, string operators, wild cards, and group operators are available for atomic expressions, and expressions can be combined by logical operators (in this example &&); logical OR (||) and negation (!) are also allowed. Combinations of simple expressions can be grouped together using parentheses. The query in figure 4 also demonstrates the use of time relations available in Q4M, e.g. '@' (temporal inclusion) and '][' (temporal contact) (see also [Mengel, 1999a]). These temporal relations are shorthand for more complex operators on the attributes of an element, and rely on a particular coding of the connection between annotation and speech, so '$a ][ $b' is the same as '($a.end == $b.start)'. If times are coded as floating point numbers, a user defined tolerance would be a useful addition here. Regular expressions can be used instead of element names to denote a range of names. Immediate and non-immediate dominance relations are provided by the '1∧' and '∧' operators, and linear precedence among children of an element is provided by the '<<' operator. Figure 5 shows an overview of the operators available in Q4M.

The result of a query is a list of tuples of nodes in the internal representation that match the query; in the example given above, these would be tuples of

| Description | Example | Operators | Explanation |
|---|---|---|---|
| Comparison of elements by the values of their attributes | | | |
| to a string | ($a.pos ~ "N") | ~ !~ | equals, does not equal |
| to a numerical value | ($a.start < 0.2) | < <= > >= == != | less, more, equal, not equal |
| relative to other values | | | |
| as a string | ($a.pos ~ $b.pos) | ~ !~ | equals, does not equal |
| as a numerical value | ($a.end > $b.end) | < <= > >= == != | less, more, equal, not equal |
| position relative to other elements | | | |
| in a hierarchy | ($a ^ $b) | n ^ m | is (nth) ancestor of (mth) child |
| in a sequence | ($a << $b) | n << m | is (mth) left neighbor (rel. to nth ancestor) |
| related to time | ($a [[ $b) | % [[ ]] [] ][ // @ | (time relations) |
| membership of a set of | | | |
| elements | ($a { $b) | { !{ {} | is member, no member, join set |
| attribute values | ($a.pos { $b.pos) | { !{ {} | is member, no member, join set |

Fig. 5. List of defined Q4M operators

($p, $s, and $w) elements. The output of Q4M is then stored as a new structure within the internal representation where it can be accessed for display to the user or be used for subsequent manipulation within the workbench. Within the MATE workbench, there is an interactive query formulation tool that assists the user when producing queries, as described in section 6. Further information about the MATE query language including a grammar and implementation details can be found in [Mengel, 1999a].

## 5 MATE stylesheets and display

In order to support flexible display and editing of corpus files, we require a flexible mapping between the *logical* structure of the data and the *display* structure. We might want to highlight certain elements which satisfy some query, only display a summary of the document, or omit certain parts of the structure. This flexibility helps in the exploration of the corpus and enables users to write specific editors for particular annotation tasks. To provide this flexibility, firstly we assume (as is fairly standard in user interface design) that the visual appearance of a displayed document can be decomposed into display objects (section 5.3) which form a hierarchical structure, such as XHTML[Pemberton, 2000] or Java display objects [Eckstein et al., 1998]. This display structure can then be described as an XML document. Formulated this way, the transformation between logical and display structures is a mapping from a directed graph to a tree. A suitable method of describing such mappings is to use a programming language based on structural recursion (see for example [Abiteboul et al., 2000, pp 101ff]).

```
<sentence>
  <det>The</det>
  <noun>cat</noun>
  <verb>sat</verb>
  <pos>on</pos>
  <det>the</det>
  <noun>mat</noun>
</sentence>
```

```
<msl:stylesheet>
  <msl:template match="($a sentence)">
    <VerticalList>
      <msl:apply-templates/>
    </VerticalList>
  </msl:template>

  <msl:template match="($a noun)">
    <TextBox colour="Red">
      <msl:apply-templates/>
    </TextBox>
  </msl:template>

  <msl:template match="($a *)">
    <TextBox colour="Black">
      <msl:apply-templates/>
    </TextBox>
  </msl:template>
</msl:stylesheet>
```

Fig. 6. An example XML file and associated stylesheet

### *5.1 Stylesheet Language and Processor*

We have defined a declarative, functional, tree-transformation language for mapping a logical document structure into a different document structure. The emerging standard in this area is XSLT, but since it was not fully defined when the workbench was designed, and still lacks some functionalities necessary to us, we decided to implement a slightly different and simpler transformation language, MATE Stylesheet Language (MSL), for our needs. MSL uses the MATE query language described above (section 4), allows the stylesheet writer to access information from the DTD, and provides a simple function language but is otherwise similar to XSLT. Transformation specifications written in XSLT or MSL are called stylesheets; each stylesheet consists of one or more templates and each template contains a query against which elements in the input document(s) are matched and a set of instructions to follow if a match is found. These instructions will often include one to recursively process the children of the matching node, hence 'structural' recursion.

Figure 6 shows an example XML file, and an example stylesheet which produces MATE display objects. It uses three templates to create display objects which will cause nouns to be displayed in red, while other words will be displayed in black. Each element in the XML file will be compared in turn against the queries in the templates until a match is found, and then the body of the template will be processed. The instruction `<msl:apply-templates/>` will cause the children of the element to be processed in their turn, and when the text children of an element are processed, a default rule adds their text to the display structure.

In order to define editing interfaces we need to be able to give the user lists of allowed element names, attribute names and allowed attribute values; this information is defined in the DTD of the annotation scheme. It is thus necessary to be able to refer to this information in stylesheets, for example to create a menu of allowable attributes. We have done this by adding new commands to the stylesheet language which provide iterators over element and attribute definitions in the DTD. An alternative approach, currently being developed by the XML Schema [Thompson et al., 2000] working group of the World Wide Web Consortium (W3C) and others, is to say that DTDs are not a sufficiently expressive data modelling language, and to seek to define an extended schema language for XML. Such a schema language can be coded in XML itself and hence the standard XSLT queries and operations can be used to access schema information.

When the stylesheet processor is run, one must specify a stylesheet written in MSL as described above, and a document or set of linked documents to transform. Normally, the stylesheet processor creates a display structure which is then processed by the display processor (section 5.3) to show something to the user. It can also be run in an alternative mode, where the input document can be transformed into an arbitrary output document structure in the internal representation, thus providing for transformations of the corpus annotations.

### *5.2 User Actions*

In order to allow the user to interact with displays created with MATE display objects, we have added certain action properties to each object which define what will happen if for example a user clicks on an object in the display. When one of these display actions occurs, code in one of the attributes of the display object describes the resulting behaviour. The code is written in a Lisp-like syntax which allows any Java method to be called. The basic idea is that the code can manipulate the internal representation and/or the display objects. In this way, user actions on the display structure can modify the internal representation. We provide a 'redisplay' method, which reruns the stylesheet on the modified part of the database and redisplays the output. A method is provided for speech playback (given a speech file and start and end times, which can conveniently be expressed as attributes of an element). Finally, methods are provided for invoking other programs or user interface components such as the query formulation window or the waveform display window. We are not entirely satisfied with the action syntax, since it is probably too complex for the average stylesheet designer, but it has enabled us to experiment with editing actions. In future work we will look at defining a set of common editing paradigms, so that stylesheet writers will be able to use these without having to know about the code that does the modifications of the database.

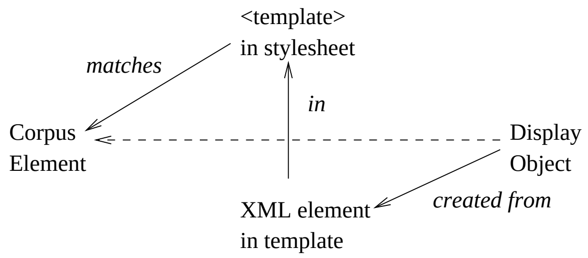


Fig. 7. Linking display objects and corpus elements

Because we want actions on the user display to have effects on the underlying corpus (i.e. we want to support editing), we need to keep back-pointers from the display objects to the parts of the corpus which they refer to. Since each display object was created by the instantiation of some template in the stylesheet, which matched against an element in the input document, this concept of back-pointer can be defined in a consistent way (as shown in figure 7).

### *5.3 Display objects*

We have defined a set of Java classes for different types of MATE display objects which are based on the Java Swing user interface classes [Eckstein et al., 1998]. At present, only a limited number of display object types have been implemented in the workbench: lists, tables, menus, sound playing, waveform display, and text. However, the workbench is set up so that it is possible to write new Java classes of display objects to provide the desired functionality. A new class must support a defined interface, which is similar to the interface required for Java user interface objects. At the time of writing, we are in discussion with other groups to extend the range of display object types that we support. For example we have recently added a display object which displays document structure as a tree using the Java interface classes. In addition, we have defined an XML DTD which describes the format of allowed display object structures to provide documentation and to allow validation of stylesheets.

## 6 Using the MATE Workbench

In this section, we provide a brief description of the MATE workbench from a user perspective. In order to use the workbench on a corpus, one needs a project directory which contains the files relating to this corpus. The workbench comes with several demonstration directories, which each contain some XML files and DTDs, stylesheets, a MATE project file which contains a list of all the files which belong to this project, and some run files which specify a stylesheet to be applied to some XML files to create a particular display.

### *6.1 Starting up the Workbench*

When the workbench is started, a file chooser menu appears, and the user can select a project which can then be run to display a particular interface, causing a list of the files referenced by the project file to appear in the right-hand window. An interface consists of one or more windows, which can each contain a speech player, or a number of panes of text. Some example interfaces are described in the next section.

Querying functions can be called in two ways: 1) the project on which the query is to be performed can be chosen from a menu in the startup window, without creating a display; 2) if a display has already been created, the project underlying the display will be queried. Both of these methods bring up a query window which interactively guides the user through the process of composing a query. The results of the query are displayed in a list, and if a display is already present, clicking on an element in the list repositions the main display on that element.

Conversion tools can be called from the startup window; we currently have two available. One converts from Entropics Xlabel format to a simple XML format, creating an element for each label (by default called "word") with start and end time attributes and content from the label. The fragment

```
0.3654   26   the

0.8234   26   cat
```

becomes

```
<word start="0.0000" end="0.3654">the</word>

<word start="0.3654" end="0.8234">cat</word>
```

The other tool performs a conversion from the BAS Partitur format [Schiel et al., 1998] into an XML format – this particular format is one of many possible XML representations.

The workbench includes an audio player program which supports the play-

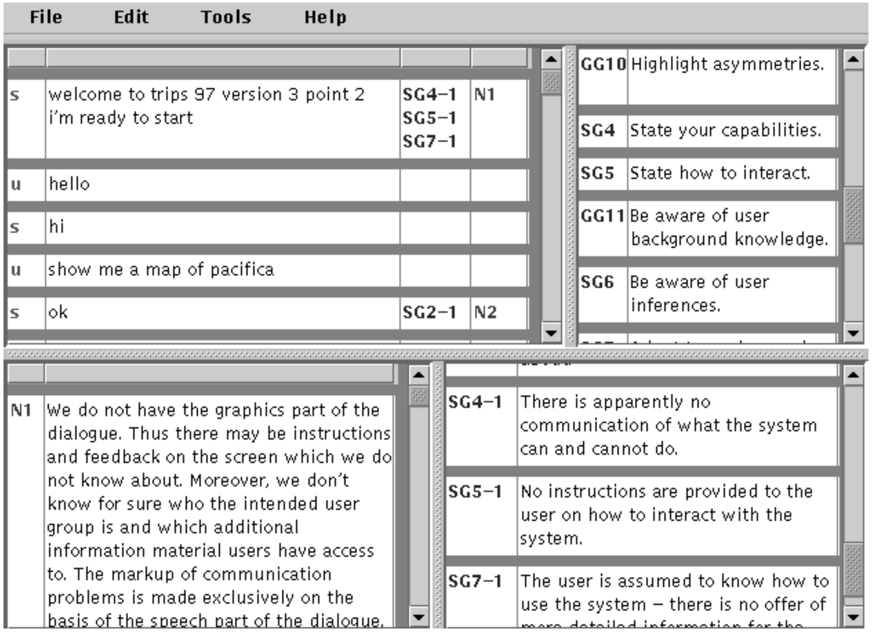


Fig. 8. A MATE interface for displaying communication problems markup

ing of $\mu$-law encoded speech files and provides limited facilities for segmenting and labelling portions of speech. Other sound file formats can be converted to this format using a publicly available conversion program such as SoX [Norskog, 1995], although a conversion program built into the workbench would be desirable. The audio player window can be started in two ways: 1) from the main tools menu, in which case the user is prompted for a filename, or 2) by the selection of an audio file in the right-hand side of the file chooser menu. This tool provides a scalable waveform display, standard playback functions and the display of intonation contours if these are provided. When a segment of speech has been selected, it is possible to use a simple transcription function, which produces an XML element with an id, start and end times, and a label, and this can be saved to a file. We do not envisage that users will want to use the MATE audio tool for large-scale transcription exercises as there are existing tools better suited to this purpose.

### *6.2 Example Interfaces*

In this section we present some examples of the MATE workbench in action. Both these examples use colours in order to make the interface clearer for the user. The display in figure 8 shows a human-machine dialogue kindly provided by TRIPS [Ferguson and Allen, 1998], which has been annotated using the MATE Communication Problems annotation scheme [Klein et al., 1998], [Mengel, 1999b]. In the top left pane, the first two columns contain a tran-

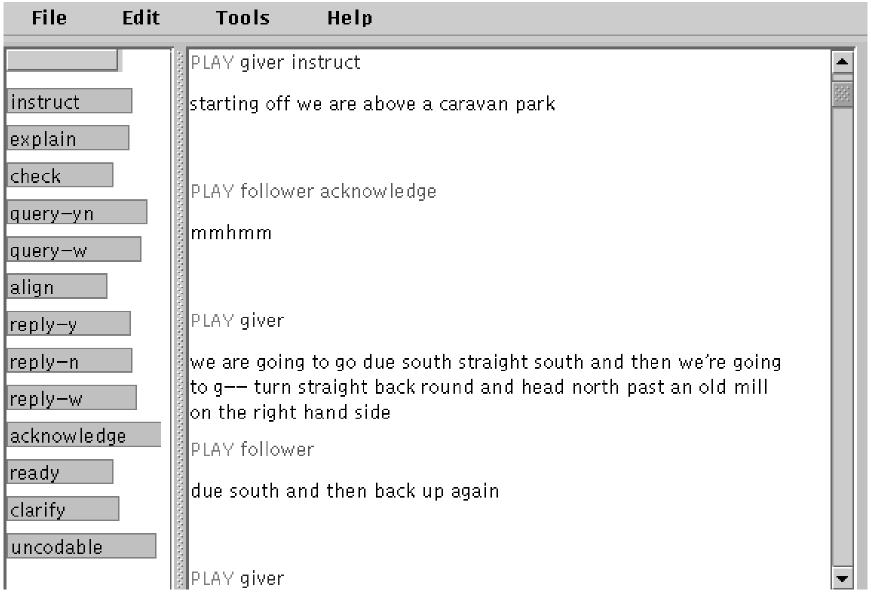


Fig. 9. A MATE interface for annotating and playing Map Task dialogue moves

scription of the dialogue, segmented into user (U) and system (S) utterances. The third column contains a list of the communication problems which occurred during each utterance, and the fourth a list of notes associated with each utterance. The bottom left pane contains the full text of all the notes pertaining to the dialogue, and the bottom right pane a full description of each communication problem. The top right pane shows a list of all the possible types of communication problem according to this annotation scheme.

Figure 9 shows the display created by running a stylesheet on a set of HCRC Map Task files. The left pane consists of a list of possible dialogue move types from the Map Task annotation scheme [Carletta et al., 1997], and the right pane shows the transcript of the dialogue, already divided into moves. The `PLAY` button above each section of text allows the user to hear the recording of the following dialogue move. This example interface allows the user to label each dialogue move with the appropriate move type from the list in the left-hand pane, by clicking on the move to be annotated, and then on the move type. This causes the chosen label to appear after the speaker name (giver or follower) in the right-hand pane. In this example, the first two moves on the screen have already been annotated, but the remainder have not.

Each of these interfaces is just one example of how one might choose to display the data. By writing another stylesheet, it is possible to create a different display which suits the needs of a different user.

## 7 Evaluation and Discussion

At the time of writing, the MATE workbench is undergoing evaluation by members of the MATE project and its panel of advisors. There is not yet a full-scale evaluation of how well the software works on a large annotation project, which is essential to fully demonstrate the validity of our approach. However, we have developed a number of different annotation schemes and stylesheets for these schemes and these are available with the workbench.

From our evaluation of these schemes and the HCRC Map Task corpus we can say that the workbench provides a flexible and powerful way of displaying complex annotations. We have found that writing stylesheets is an easy way to tailor displays to focus attention on particular aspects of the data. The query language allows one to extract parts of the corpus based on complex criteria, and is also central to the power of the stylesheet transformations. The action language provides us with the ability to create and modify annotations, although further work is needed to make writing stylesheets that contain actions easier. More work is needed to document the interface to the database to allow other code to access it.

### *7.1 Discussion*

There are a number of ideas underlying the structure of the MATE workbench, which we believe are important qualities for any speech annotation system to have.

(1) Because of the similarities between speech annotations and other kinds of annotation, one should situate the development of speech annotation tools into the broader context of XML editors and browsers. The same is true for query languages. This is not to deny that handling speech requires specialised software, as does the annotation of video data.
(2) As linguistic annotations of a corpus become more complex, for example by including many different types or versions of annotation, it becomes very useful to have flexible options for displaying the data. If one can select and highlight particular aspects of the data, it is then possible to use browsing as a way of building intuitions about the structures in the data and as an aid to the formulation of hypotheses which can later be confirmed using the query language. Similarly, the ability to define editors tailored to particular annotation tasks can speed up annotation creation or checking. We suggest that the use of a high level transformation language to provide a flexible link between logical structures and display

structures is the correct way to provide such facilities.

(3) The system design should be reflexive – definitions of user interfaces, query results, and corpus descriptions should all be in the same format and expressible in the same data model. If query results are represented as documents, they can be saved as files (becoming part of the corpus) and the stylesheet language can then be used to define how they will be displayed. If stylesheets or corpus descriptions (whether DTDs, XML Schemas or textual comments) are represented as documents, we can use the workbench to write specialised editors to create them, as well as use the information (such as default and allowed values) they contain to write stylesheets to display the annotated corpus.

(4) In order to handle the complexity of linguistic annotations the data model needs to extended from trees to general graphs. The query and transformation languages used should reflect this data model. This point is generally accepted, as demonstrated by the other papers in this volume, although there are differences between the proposed models. For example, the three data types implemented in [Taylor et al., 2000], can also be represented in the MATE model, although Taylor's implementation may be more efficient. Another useful feature of Taylor's data model is the provision of calculated or constrained attribute values. The data model proposed in [Bird and Liberman, 2000a] would seem to be useful for cases where there is massive ambiguity or overlap between annotation segments such as word lattices generated by speech recognisers, where the MATE method of hyperlinks could become rather verbose. By carefully distinguishing between abstract data model, implementation and interchange formats, it should be possible to achieve convergence between these different proposals.

(5) The display processor should be extensible, so that it is easy to add new display options, for example to add tree/graph displaying capabilities.

### *7.2 Future directions*

The transformation language and the data model need to be extended to encompass dynamic interfaces and the updating of the document structure, which are necessary to support the editing of corpus annotations.

We have chosen an architecture where entire files are loaded into memory and processed as a group. The alternative would be to provide a streaming interface where larger files were read and processed a section at a time, and the extent of a section would be defined by a query on the XML structure of the file. If we were to re-implement the workbench, we would reconsider using the XSLT query and transformation languages (once they become defined as standards). However, they would need to be extended to support DTD access, hyperlinked documents and the more extensive capabilities of the MATE query language before we could do this. In addition, we would consider using a separate database program to support the internal representation, which would provide faster access to larger corpora.

### *7.3 Availability*

At the time of writing, the MATE workbench is undergoing evaluation by members of the MATE advisory panel. To get further documentation on the workbench and to obtain a copy of the software, please consult the MATE workbench web page at `http://www.ltg.ed.ac.uk/software/mate/`.

## 8 Acknowledgements

The work described here was funded by the European Union (MATE project: Telematics LE4-8370). We wish to thank our respective institutes for their support. We would also like to thank Steven Bird and the anonymous reviewers for helpful comments on the content and style of this paper. We are grateful to TRIPS, The Rochester Interactive Planning System for providing a dialogue which has been annotated using the MATE annotation schemes.

## References

[Abiteboul et al., 2000] Abiteboul, S., Buneman, P., and Suciu, D. (2000). *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann, San Francisco, California.

[Barras et al., 2000] Barras, C., Geoffrois, E., Wu, Z., and Liberman, M. (2000). Transcriber: development and use of a tool for assisting speech corpora production. *This volume.* See also www.etca.fr/CTA/gip/Projets/Transcriber.

[Bird and Liberman, 2000a] Bird, S. and Liberman, M. (2000a). A formal framework for linguistic annotation. *This volume.*

[Bird and Liberman, 2000b] Bird, S. and Liberman, M. (2000b). Linguistic annotation resources. Linguistic Data Consortium. www.ldc.upenn.edu/annotation.

[Boersma and Weenik, 2000] Boersma, P. and Weenik, D. (2000). Praat, a system for doing phonetics by computer, version 3.4. Technical Report 132, Institute of Phonetic Sciences of the University of Amsterdam. www.praat.org.

[Bray et al., 1998] Bray, T., Paoli, J., and Sperberg-McQueen, C. M. (1998). Extensible Markup Language. World Wide Web Consortium. www.w3.org/TR/REC-xml.

[Carletta et al., 1997] Carletta, J., Isard, A., Isard, S., Kowtko, J., Doherty-Sneddon, G., and Anderson, A. (1997). The reliability of a dialogue structure coding scheme. *Computational Linguistics*, 23:13–31.

[Cassidy and Harrington, 2000] Cassidy, S. and Harrington, J. (2000). Multi-level annotation of speech: An overview of the Emu speech database management system. *This volume.* See also www.shlrc.mq.edu.au/emu.

[Clark, 1999] Clark, J. (1999). XSL Transformations (XSLT). World Wide Web Consortium. www.w3.org/TR/xslt.

[Cunningham et al., 1996] Cunningham, H., Wilks, Y., and Gaizauskas, R. J. (1996). GATE – a general architecture for text engineering. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 1057–1060, Copenhagen, Denmark. See also www.dcs.shef.ac.uk/research/groups/nlp/gate.

[Day et al., 1997] Day, D., Aberdeen, J., Hirschman, L., Kozierok, R., Robinson, P., and Vilain, M. (1997). Mixed-initiative development of language processing systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington D.C. Association for Computational Linguistics. See also www.mitre.org/technology/alembic-workbench.

[DeRose et al., 1999] DeRose, S., Jr., R. D., and Maler, E. (1999). XML Pointer Language (XPointer). World Wide Web Consortium. www.w3.org/TR/xptr.

[DeRose et al., 2000] DeRose, S., Maler, E., Orchard, D., and Trafford, B. (2000). XML Linking Language (XLink). World Wide Web Consortium. www.w3.org/TR/xlink.

[Deutsch et al., 1998] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu, D. (1998). XML-QL: A query language for xml. World Wide Web Consortium. www.w3.org/TR/NOTE-xml-ql.

[Eckstein et al., 1998] Eckstein, R., Loy, M., and Wood, D. (1998). *Java Swing*. O'Reilly & Associates, Sebastopol, CA.

[Entropic, 1996] Entropic (1996). *Waves+ Manual*. Entropic Research Laboratory Inc. www.entropic.com.

[Ferguson and Allen, 1998] Ferguson, G. and Allen, J. F. (1998). TRIPS: An intelligent integrated problem-solving assistant. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 567–573, Madison, WI.

[Goldman et al., 1999] Goldman, R., McHugh, J., and Widom, J. (1999). From semistructured data to XML: Migrating the Lore data model and query language. In *Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99)*, Philadelphia, Pennsylvania. See also www-db.stanford.edu/lore.

[Grisham, 2000] Grisham, R. (2000). Tipster text architecture concept. Technical report, Defense Advanced Research Projects Agency. www-nlpir.nist.gov/related_projects/tipster/docs/con112.doc.

[Grugman and Russel, 1999] Grugman, H. and Russel, A. (1999). Eudico: European distributed corpora project. Max Planck Institute for Psycholinguistics, Nijmegen. www.mpi.nl/world/tg/lapp/eudico/eudico.html.

[Heid and Mengel, 1999] Heid, U. and Mengel, A. (1999). A query language for research in phonetics. In *Proceedings of the International Congress of Phonetic Sciences (ICPhS99)*, San Francisco, California.

[Ide et al., 2000] Ide, N., Priest-Dorman, G., and Bonhomme, P. (2000). Corpus encoding standard. Expert Advisory Group on Language Engineering Standards (EAGLES). www.cs.vassar.edu/CES.

[Ide and Véronis, 1995] Ide, N. and Véronis, J., editors (1995). *The Text Encoding Initiative: Background and Context*. Dordrecht: Kluwer. See also www.tei-c.org.

[Isard et al., 1998] Isard, A., McKelvie, D., and Thompson, H. S. (1998). Towards a minimal standard for dialogue transcripts: A new SGML architecture for the HCRC Map Task corpus. In *Proceedings of the 5th International Conference on*

*Spoken Language Processing, ICSLP98*, Sydney, Australia. www.ltg.ed.ac.uk/Papers/∼dmck/icslp98.ps.

[Jacobson et al., 2000] Jacobson, M., Michailovsky, B., and Lowe, J. B. (2000). Linguistic documents synchronizing sound and text. *This volume.* See also lacito.vjf.cnrs.fr/ARCHIVAG/ENGLISH.htm.

[Jelliffe, 2000] Jelliffe, R. (2000). *Schematron.* Academica Sinica Computing Centre. www.ascc.net/xml/resource/schematron/schematron.html.

[Klein et al., 1998] Klein, M., Bernsen, N. O., Davies, S., Dybkjaer, L., Garrido, J., Kasch, H., Mengel, A., Pirelli, V., Poesio, M., Quazza, S., and Soria, C. (1998). MATE deliverable 1.1: Supported coding schemes. mate.nis.sdu.dk/about/deliverables.html.

[LeMaitre et al., 1996] LeMaitre, J., Muriasco, E., and Rolbert, M. (1996). SgmlQL, a language for querying SGML documents. In *Proceedings of the 4th European Conference on Information Systems (ECIS'96)*, Lisbon, Portugal. See also www.univ-tln.fr/∼gect/simm/SgmlQL.

[Marchiori, 1998] Marchiori, M. (1998). QL'98 - the Query Languages Workshop. World Wide Web Consortium. www.w3.org/TandS/QL/QL98.

[Mengel, 1999a] Mengel, A. (1999a). *Manual for Q4M.* Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart. www.ims.uni-stuttgart.de/projekte/mate/q4m.

[Mengel, 1999b] Mengel, A. (1999b). MATE deliverable 2.1: DTDs and notes on schemas. mate.nis.sdu.dk/about/deliverables.html.

[Murata and Robie, 1998] Murata, M. and Robie, J. (1998). Observations on structured query languages. In Marchiori, M., editor, *Proceedings of QL'98 - The Query Languages Workshop*, Boston, Massachussets. www.w3.org/TandS/QL/QL98/pp/murata-san.html.

[Norskog, 1995] Norskog, L. (1995). SoX sound file format converter. www.spies.com/Sox.

[Pemberton, 2000] Pemberton, S. (2000). XHTML 1.0: The Extensible Hypertext Markup Language. World Wide Web Consortium. www.w3.org/TR/xhtml1.

[Schiel et al., 1998] Schiel, F., Burger, S., Geumann, A., and Weilhammer, K. (1998). The Partitur format at BAS. In *Proceedings of the First International Conference on Language Resources and Evaluation*, Granada, Spain.

[Sjölander, 2000] Sjölander, K. (2000). The Snack sound extension for Tcl/Tk. Department of Speech Music and Hearing, Royal Institute of Technology, Stockholm. www.speech.kth.se/snack.

[Taylor et al., 2000] Taylor, P., Black, A. W., and Caley, R. (2000). Heterogenous relation graphs as a formalism for representing linguistic information. *This volume.*

[Taylor et al., 1999] Taylor, P., Caley, R., Black, A., and King, S. (1999). Edinburgh speech tools library. Centre for Speech Technology Research, University of Edinburgh. www.cstr.ed.ac.uk/projects/speech_tools.

[Thompson et al., 2000] Thompson, H. S., Beech, D., Maloney, M., and Mendelsohn, N. (2000). XML Schema part 1: Structures. World Wide Web

Consortium. www.w3.org/TR/xmlschema-1.

[Vatton et al., 1999] Vatton, I., Guétari, R., Kahan, J., and Quint, V. (1999). Amaya – W3C's editor/browser. World Wide Web Consortium. www.w3.org/Amaya.

[Wood et al., 2000] Wood, L., Hors, A. L., Apparao, V., Cable, L., Champion, M., Davis, M., Kesselman, J., Hégaret, P. L., Pixley, T., Robie, J., Sharpe, P., and Wilson, C. (2000). Document Object Model (DOM) level 2 specification. World Wide Web Consortium. www.w3.org/TR/DOM-Level-2.