The MATE Workbench - A tool for annotating XML corpora

Amy Isard[1], David McKelvie[1], Andreas Mengel[2], Morten Baun Møller[3]

1. Language Technology Group, Division of Informatics, Edinburgh University;
2. Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart;
3. Natural Interactive Systems Laboratory, Odense University.

Abstract

This paper describes the design and implementation of the MATE workbench, a program which provides support for flexible display and editing of XML annotations, and complex querying of a set of linked files. The workbench was designed to support the annotation of XML coded linguistic corpora, but it could be used to annotate any kind of data, as it is not dependent on any particular annotation scheme. Rather than being a general purpose XML-aware editor it is a system for writing specialised editors tailored to a particular annotation task. A particular editor is defined using a transformation language, with suitable display formats and allowable editing operations. 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.

1. Introduction

The annotation or markup of files with linguistic or other complex information usually requires either human coding or human correction of automatic coding. This annotation can be a time consuming and tedious process, yet it is a necessary one for the creation or analysis of corpora and the provision of data necessary for training automatic annotation programs, such as parsers, speech recognisers, or named entity recognisers. In order to support the creation and use of such annotations, specialised editing, retrieval and display programs are required. However, the creation of these programs can in itself be a complex process. In the MATE workbench, we have attempted to provide a parametrised tool which can be specialised to provide these functions on any annotation scheme capable of being expressed in XML [Bray, 1998].

An annotation tool which is specialised for 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 vastly preferable to a general purpose editor. However, this normally requires writing individual editors for each task. In the MATE workbench the corpus designer can write rule based stylesheets (based on the XSLT transformation language [Clark, 1999]) which describe how the corpus is presented to the annotator and what editing actions are allowed. The Workbench provides a number of pre-defined stylesheets for use with particular nnotation schemes developed within the MATE project, but its major strength lies in the fact that the stylesheet language is sufficiently high-level for writing stylesheets to be significantly easier than writing an editor from scratch.

The major features of the MATE workbench are:

- An internal database - using arbitrary XML as an interchange format, extended to cover multiple hierarchies and arbitrary directed graphs using hyperlinks [DeRose, 1999] or ID/IDREF [Goldfarb, 1990] pointers between elements. This extension from trees to graphs is required to allow XML to represent more complex data.
- A query language which is tailored to this internal representation. This language returns tuples instead of single elements (as in the XSLT query language). The architecture allows us to add new structure to the database by evaluating a query.

- A transformation language and processor that goes beyond XSLT in some respects.
- A display and editing engine for displaying to the user and enabling editing actions.

2. Review

The idea of using SGML(XML) as a annotation standard for NLP goes back to the work of the Text Encoding Initiative (TEI) [Sperberg-McQueen, 1994]. Using a database as a central resource for natural language processing was developed by the Gate project [Cunningham, 1997]. Data models related to XML and associated query languages were developed by a number of groups including the Lore project [Abiteboul, 1997] and SGMLQL [Le Maitre, 1996]. The idea of document transformation using structurally recursive rules comes from the work of the DSSSL standard and XSLT working group. There have been a number of tools developed for the support of linguistic annotation, for example, Transcriber, Annotag, NB, EMU, Alembic (see below). What the MATE project is attempting to do is to bring these strands of work together to provide a generic annotation tool, which supports non-hierarchic markup (particularly necessary for speech due to overlapping annotation schemes, multiple overlapping speakers). Stylesheet transformation languages have so far largely been used to produce static documents, but in this project we extend this to create active editable displays (see also the Amaya web editor [Vatton, 1999] for a similar approach). An extensive list of tools, with links, can be found at the Linguistic Data Consortium's Linguistic Annotation web site [Bird, 2000]. We will give an overview of some of these tools.

There are a number of systems which focus on higher level annotation. The Alembic workbench [Day, 1999] 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, 1996], and can also output annotations in an SGML format. It does not provide facilities for listening to or manipulating speech signals. The Lacito Archivage tool [Michailovsky, 1999] displays recordings and annotations made by Lacito members over the last thirty years. The annotations are stored as XML, and XSL stylesheets are used to provide multiple views of the same data. Displays are made using Java applets, and can be shown on any standard web browser.

There are systems which aim to bring together existing tools. GATE [Cunningham, 1997] supports modules written in any programming language. It uses the Tipster architecture internally, but also supports SGML input and output. 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 [Brugman, 1999] is platform independent, written in Java, and is usable 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.

There are several tools which provide speech signal manipulation capabilities far more sophisticated than those currently available in the MATE workbench, (e.g. Emu [Cassidy, 2000] , Praat [Boersma, 2000] and Transcriber [Barras, 2000]). 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 some form of XML. A tool for converting Entropic Xlabel format to XML is provided with the MATE workbench, see (section 6).

Alembic and Lacito both provide similar functionality to some 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. 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. System Architecture

3.1 XML as data format

XML has proved to be a widely used and effective format for annotating text and speech corpora [Sperberg-McQueen, 1994] [Ide, 1999]. Its flexible hierarchical structure matches well with many areas of linguistic theory. The MATE workbench uses XML as its input/output format, and uses a similar internal data model. However, the strictly hierarchic 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 overlap, for example prosody and syntax. One way to indicate this non-hierarchical structure in XML is by the use of standoff annotation [Isard, 1998]. Linking between elements is done by means of a distinguished *href* attribute of elements, which uses a subset of the XPOINTER proposal [DeRose, 1999] to point to arbitrary elements in the same or different files. Such attributes are often called hyperlinks. This extended data model allows us to represent overlapping or crossing annotations. For example, the following non well-formed XML 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 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)”/>
```

The representation of this in the MATE database (see section 3.3) is shown in figure 1. Note that this could all be in one file or separate files for the two hierarchies. Standoff annotation (where hyperlinks refer to elements in a 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 MATE workbench is designed to work on annotation corpora coded in XML. It is not restricted to a particular annotation scheme, described in XML by a Document Type Definition (DTD).

The architecture of the MATE workbench consists of the following major components:

- An internal database which is an in-memory representation of a set of hyperlinked XML documents (section 3.3). There are functions for loading and outputting XML files into and out of the database;

- A query language and processor which are used to select parts of this database for subsequent display or processing (section 4);

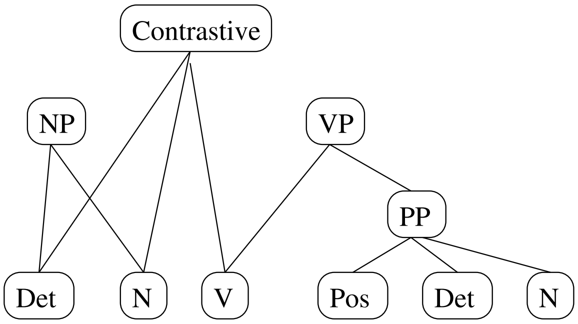

Figure 1: Internal representation of example with crossing annotations

- 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 (section 5);
- A display processor which is responsible for handling the display and editing actions (section 5.4). This takes the display object output of a stylesheet transformation and shows it to the user;
- A user interface which handles file manipulation and tool invocation (section 5).

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). 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 specialism in software design as elsewhere in life. Specialised facilities in annotation tools (e.g. automatic annotation) will always require writing specialised software. However, we think that it is worthwhile building 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, although these also have their place. 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 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 (extension to typed attributes would be possible) and an ordered 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 is used to decompose the graph into a disjoint set of trees. 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, for example, there is no reason why one could not have a node of type *or* whose children represented alternative descriptions of the same segment of speech.

The representation of this data model in XML 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

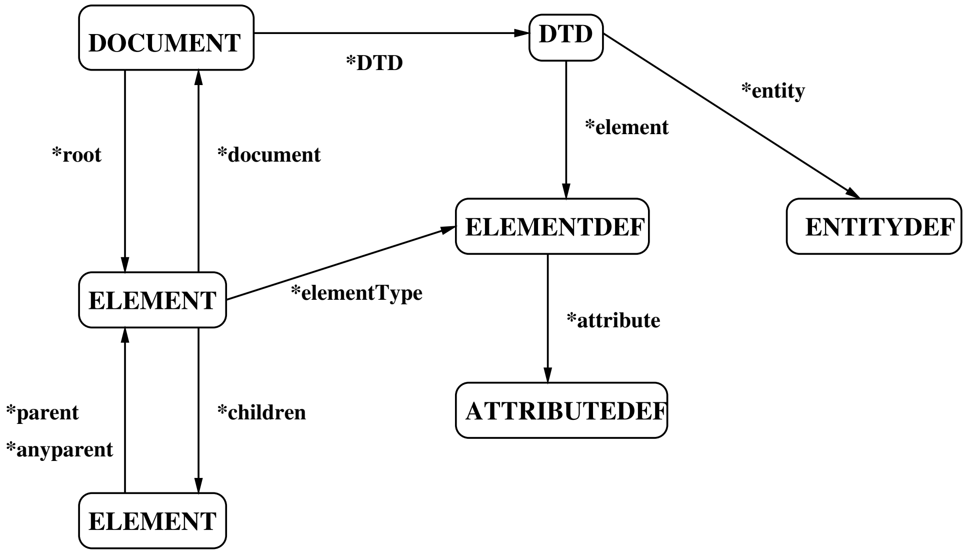

Figure 2: A partial schema for the MATE internal representation

name of the XML element (or "#PCDATA" for text). Attributes are copied over (taking default values from the 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 forest of trees. As a final stage of loading, hyperlinks between these trees are constructed as additional (non-distinguished) parent-child links. This gives the ability to represent arbitrary directed graphs in XML. 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. This does not change the contents of the internal representation once these files are read back in, as the loading software can recognise and ignore the dummy elements.

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 have values which are lists of other nodes in the internal representation, for example the **children* and **parent* properties. As an extension to the standard XML document model [Wood, 1998], DTDs are also represented as objects in the database. The types of nodes and their interrelationships are shown in figure 2. This representation of the database makes it easy for the query language (section 4) to ask general questions about the annotations.

Most uses of this data model for describing corpora so far have resulted in graphs which are acyclic, but we are not yet sure whether we want to enforce this restriction. Our data model is similar to that proposed by the Lore project [Goldman, 1999], although we use XPOINTER attributes rather than IDs for hyperlinking, since that allows us to link across files without assuming a single ID name space 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 cross links between the files represented by hyperlinks, so for example there is currently no way to ensure that all hyperlinks from utterance annotations are to 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. One could currently 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 to display them in an error window.

An important point in the workbench design is that it is reflexive, i.e. all information about the system

(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 ][ $w) &&`	<`sent`> precedes <word> AND
(7):	`($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*”

Figure 3: An example Q4M query.

is kept in a similar format in the internal representation. For example, both stylesheets (which describe the appearance of the user interface) and results of queries are stored in the same format. This has the advantage that, for example, we could use the workbench to provide a stylesheet editor.

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 in the corpus.

4. The MATE Query Language Q4M

XML is now being used for purposes and domains previously covered by relational databases. Because XML is strongly hierarchical and its structure is flexible (e.g. optional or repeated elements), coding it in a relational database can be a relatively expensive operation. This has led to a large number of proposed XML query languages [Marchiori, 1998]. Similar work was also done in the database community on developing query languages for data models which were more flexible than relational models, such as the work on ‘semi-structured’ databases by the Lore project [Goldman, 1999]. We nonetheless chose to develop our own query language and processor (Q4M). The reasons were that; (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 graph and not just a tree structure.

A query language appropriate for the data model requires the following properties (see [Murata, 1998] for general considerations of query languages in the XML world). Firstly, XML constructs like element, attribute and attribute value must be accessible to the query language. Secondly, 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 this order. This is not always supported, as in early versions of the ‘semi-structured’ data model. Thirdly, we also want to be able to return tuples of elements, i.e. combinations of elements with specific properties, pairs of elements with comparable properties, elements in a hierarchical relation and so on. This feature goes beyond some other XML query languages (such as the one defined in XSLT) which are restricted to returning lists of elements. One example would be a query such as the following.

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

Figure 3 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, wildcards, and group operators are available for atomic expressions.

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 into complex brackets.

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

Figure 4: List of defined Q4M operators

The query in figure 3 also demonstrates the use of time relations available in Q4M, e.g. '@' (temporal inclusion) and '][' (temporal contact) (see also [Mengel, 1999a], [Heid, 1999]). The 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, for example "$a][$b" is the same as "($a.end == $b.start)". Figure 4 is 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 ($p, $s, and $w) elements. The output of Q4M is 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 details of 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. For example, we might want to highlight certain elements which satisfy some query or 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 knowledgeable users to write specific annotation 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 5.4 which form a hierarchical structure, for example XHTML or Java display objects. 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'.

5.1 Stylesheet Language

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 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 defined above (section 4), but is otherwise similar to XSLT. Transformation specifications written in XSLT or MSL are often called stylesheets. Each stylesheet consists of one or more templates; 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

```
<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>
```

Figure 5: An example XML file and associated stylesheet

found. These instructions will often include one to recursively process the children of the matching node, hence 'structural' recursion.

Figure 5 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, they are printed out.

5.2 Stylesheet Processor

When the stylesheet processor is run, a document or set of linked documents in the internal representation is processed along with a stylesheet written in MSL, described in section 5.1. Normally, the Stylesheet Processor outputs a display structure as described above, which is then processed by the Display Processor to show something to the user. It can also be run in an alternative mode, in which case the input document can be transformed into an arbitrary output document structure in the internal representation, thus providing for transformations of the corpus annotations.

5.3 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. At present, only a limited number of display object types have been implemented in the workbench, basically lists, tables, menus, sound playing, waveform display, and text. So, for example, it is not currently possible to display structure as a parse tree. 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. We are currently in discussion with other groups to extend the range of display object types that we support.

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.

5.4 Display Actions

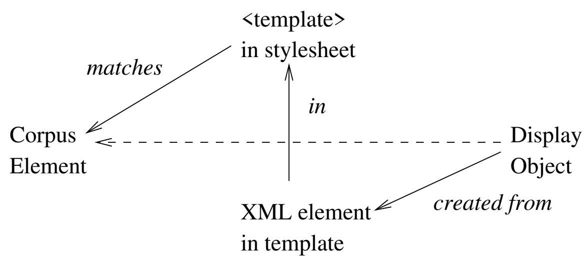

Figure 6: Linking display objects and corpus elements

In order to allow the user to interact with displays created with MATE display objects, we have added certain action properties to each object. These define, for example, what will happen if 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 defines what is to happen. 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, thus 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. We are not entirely happy with this action syntax, since it is probably too complex for the average stylesheet designer. However it has let us 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. In order to define editing interfaces we need the ability 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 schema. 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 iteration over element and attribute definitions in the DTD. With the development of XML Schemas [Thompson, 1999] to replace DTDs, this may become possible in XSLT in the future.

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 6).

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, you need a project directory which contains the files relating to this corpus. The workbench comes with several demo 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.

When the workbench is started, a File chooser menu appears (figure 7), and the user can select a project which can then be run to display a particular interface. In this example, the user has chosen the comprobs.mp MATE project file in the left-hand window, which has caused a list of the files referenced by the project file to appear in the right-hand window: one stylesheet, four XML files and a run file. An interface consists of one or more windows, which can 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

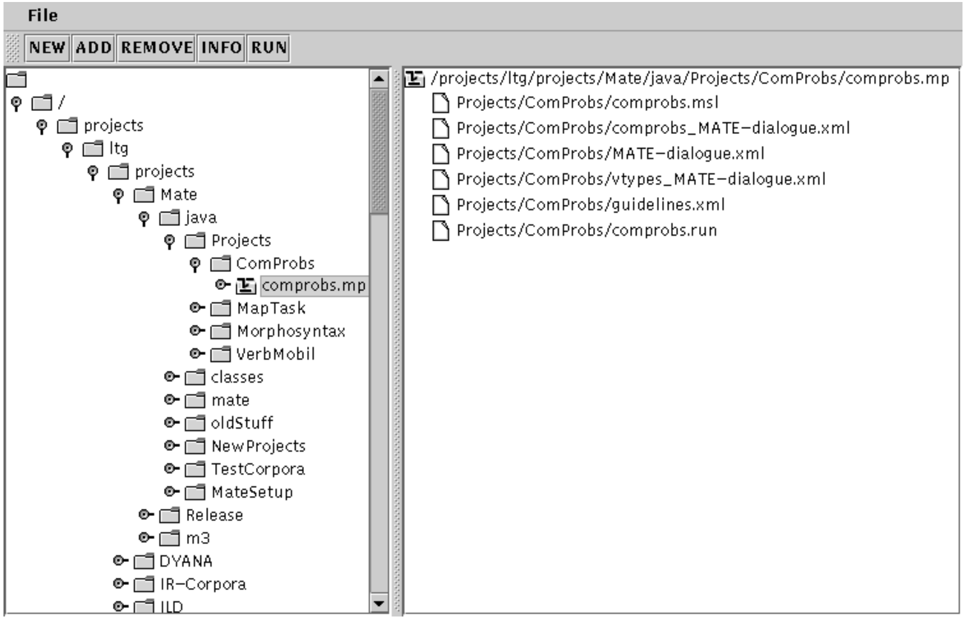

Figure 7: MATE file menu chooser

be chosen from a menu in the startup window, without creating a display; 2) If a display has already been created, the project queried will be the one used in creating the display. 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 takes you to the position of that element in the main display.

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 other tool performs a conversion from the BAS Partitur format into an XML format – this particular format is one of many possible XML representations.

6.1 Examples

In this section we present examples of the MATE workbench in action. In figure 8, we see the display which is created by clicking on the RUN button in figure 7. This display shows a human-machine dialogue annotated using the MATE Communication Problems annotation scheme [Mengel, 1999b]. In the top left pane, the first two columns contain a transcription 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 the 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, 1996]. 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. Both the

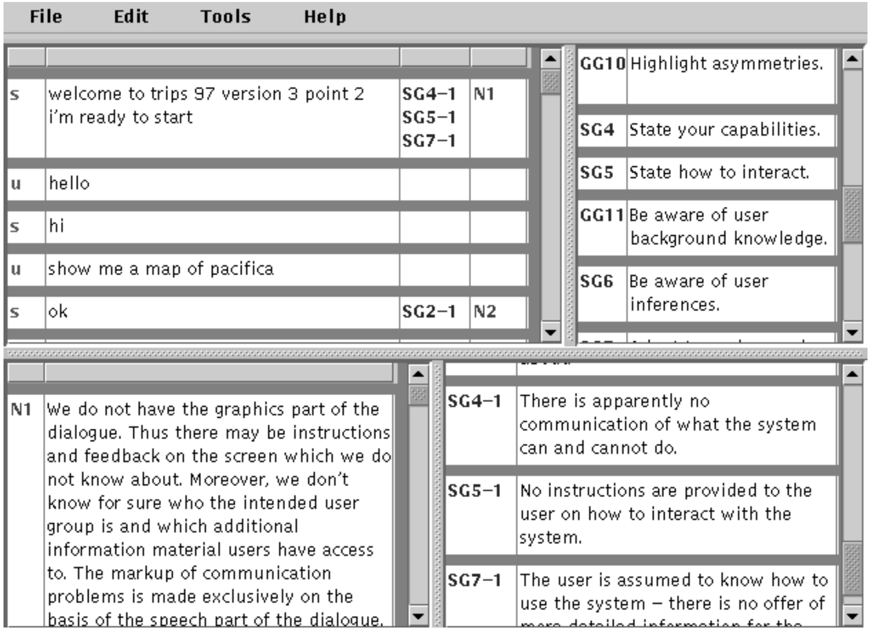

Figure 8: A MATE interface for displaying communication problems markup

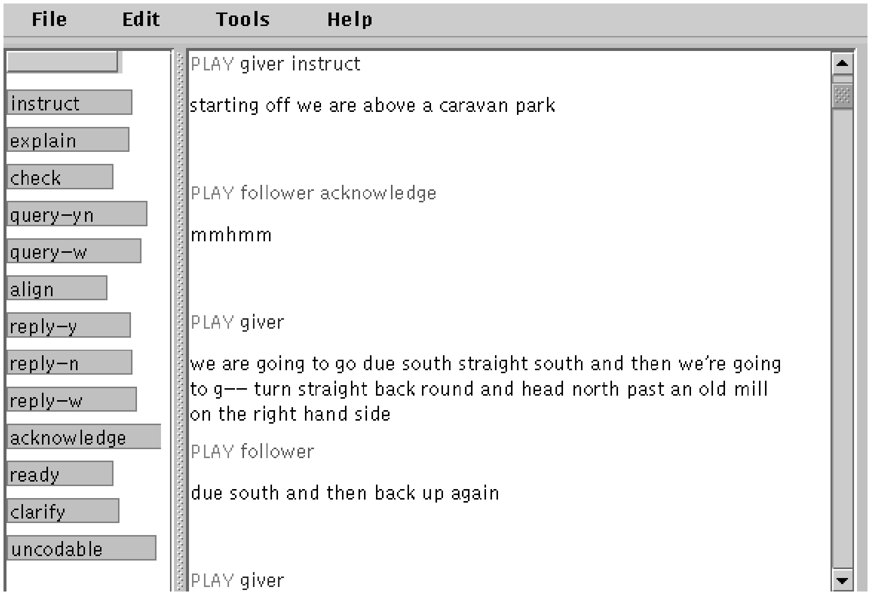

Figure 9: A MATE interface for annotating and playing Map Task dialogue moves

examples described above use colours in order to make the interface clearer for the user.

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

7.1 Evaluation

The MATE workbench is currently 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 obviously essential to fully demonstrate the validity of our approach. However, we have developed a number of different annotation schemes and stylesheets for these schemes (which are available with the workbench).

From our evaluation on 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 was 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 vital 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 to write. More work is needed on documenting the interface to the database to allow other code to access it, as are further extensions to the range of display types supported.

7.2 Discussion

We believe that the MATE project has developed a prototype annotation tool which demonstrates a feasible approach to making specialised editors easier to develop and to use. This claim will be tested in future work at Edinburgh where we will use the workbench to annotate a corpus of telephone dialogues. The system is based on developing standards in the XML community. There are a number of ideas which underlie the structure of the MATE workbench, which we believe are important.

1) For flexibility of display and the easy definition of editors tailored to a particular annotation task, one should use a high level transformation language to provide a flexible link between logical structures and display structures.

2) 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.

3) In order to handle the complexity of linguistic annotations and a reflexive system design one needs to extend the data model away from trees towards general graphs. The query and transformation languages used should reflect this data model.

4) The display processor should be extendible, so that it becomes easy to add new display options, for example to add tree/graph displaying capabilities.

7.3 Future directions

However there are a number of aspects of the prototype system which it would be useful to readdress if we were to re-implement the workbench. Firstly, now that the XSLT language is becoming stable, it would be sensible to address the issue of using the standard XSLT transformation language instead of our cut-down version of it. This would involve (a) using the XPATH query language in stylesheets, (b) addressing how handle hyperlinking in stylesheets, and (c) using XMLSchema to enable access to DTD-type information in stylesheets. An alternative (more speculative) approach would be to keep the MATE query language and extend stylesheet templates to work with tuple-returning queries. Secondly, to address the issue of efficiency and allow the workbench to be used with much larger files. This could involve reimplementing the workbench to use an external database. 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 (where the definition of a section would be defined by a query on the XML structure of the file).

Thirdly, extend the display objects so as to be able to use any Java (Swing) user interface component in stylesheets and in displays. This would involve writing a schema (DTD or otherwise) for the Java classes, which would be a useful exercise in its own right. Finally, the transformation language and the data model need to be extended to encompass dynamic interfaces and the updating of the document structure. These are necessary to support the editing of corpus annotations. The two directions for future work are making the language for defining editing actions easier to use, and allowing display structures to be created dynamically, i.e. popup menus.

7.4 Availability

The MATE workbench is currently undergoing evaluation by members of the MATE advisory panel. If you wish to obtain a copy of the software for evaluation, please look at the web site `http://mate.nis.sdu.dk` for details of how to access the software. The source code will be made available under a GNU-style license later this year.

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.

References

[Abiteboul, 1997] Abiteboul, S., Quass, D., McHugh, J., Widom, J. & Wiener, J.L. (1997). The Lorel Query Language for Semistructured Data, *Journal on Digital Libraries*, 1(1).

[Barras, 2000] Barras, C., Geoffrois, E., Wu, Z. & Liberman M. (2000). Transcriber: development and use of a tool for assisting speech corpora production, *Speech Annotation and Corpus Tools, a special issue of Speech Communication* (to appear). See also `http://www.etca.fr/CTA/gip/Projets/Transcriber`.

[Bird, 2000] Bird, S. & Liberman, M. (2000). Linguistic Annotation Resources, Linguistic Data Consortium, University of Pennsylvania. See `http://www.ldc.upenn.edu/annotation`

[Boersma, 2000] Boersma, P. & Weenink, D. (2000). Praat: doing phonetics by computer, Institute of Phonetic Sciences, University of Amsterdam. See `http://www.fon.hum.uva.nl/praat`

[Bray, 1998] Bray, T., Paoli, J. & Sperberg-McQueen, C. M. (editors) (1998). Extensible Markup Language (XML) 1.0, World Wide Web Consortium. See `http://www.w3.org/TR/REC-xml`.

[Brugman, 1999] Brugman H. & Russel A., EUDICO: European Distributed Corpora Project (1999). Max Planck Institute for Psycholinguistics, Nijmegan. See `http://www.mpi.nl/world/tg/lapp/eudico/eudico.html`

[Carletta, 1996] Carletta, J. et al. (1996). HCRC dialogue structure coding manual. HCRC Technical Report.

[Cassidy, 2000] Cassidy, S. & Harrington, J. (2000). Multi-level annotation of speech: An overview of the Emu Speech Database Management System, in *Speech Annotation and Corpus Tools, a special issue of Speech Communication* (to appear). See also `http://www.shlrc.mq.edu.au/emu`.

[Clark, 1999] Clark, J. (editor) (1999). XSL Transformations (XSLT), Version 1.0, World Wide Web Consortium. `http://www.w3.org/TR/xslt`

[Cunningham, 1997] Cunningham, H., Humphreys, K., Gaizauskas R. J. & Wilks, Y. (1997). Software Infrastructure for Natural Language Processing. In *5th Conference on Applied Natural Language Processing*, Washington. See also `http://www.dcs.shef.ac.uk/research/groups/nlp/gate`

[Day, 1999] Day, D. (2000). Alembic Workbench Project, The Mitre Corporation. See `http://www.mitre.org/technology/alembic-workbench`

[DeRose, 1999] DeRose S., Daniel R. & Maler E. (eds) (1999). XML Pointer Language (XPointer), World Wide Web Consortium. See `http://www.w3.org/TR/WD-xptr`.

[Goldfarb, 1990] Goldfarb, C. M., (1990). The SGML Handbook. Clarendon Press.

[Goldman, 1999] Goldman R., McHugh J. & 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.

[Grisham, 1996] Grisham, R., Gaizauskas, R., Cunningham, H. & Zajac, R. (1996). TIPSTER Text Program, Defense Advanced Research Projects Agency. See `http://www.itl.nist.gov/iaui/894.02/related_projects/tipster`

[Heid, 1999] Heid, U. & Mengel. A. (1999). A Query Language for Research in Phonetics, in *ICPhS 99 (International Congress of Phonetic Sciences)*, San Francisco.

[Ide, 1999] Ide, N. (co-ordinator) (1999). Corpus Encoding Standard, Expert Advisory Group on Language Engineering Standards (EAGLES). See `http://www.cs.vassar.edu/CES/CES1.html`

[Isard, 1998] Isard, A., McKelvie, D. & 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. `http://www.cogsci.ed.ac.uk/~dmck/icslp98.ps`

[Le Maitre, 1996] Le Maitre, J., Murisasco, E. & Rolbert, M. (1996). SgmlQL, a language for querying SGML documents. In *Proceedings of the 4th European Conference on Information Systems (ECIS'96) (pp. 75-89)*. Lisbon. See also `http://www.univ-tln.fr/ gect/simm/SgmlQL/`

[Marchiori, 1998] Massimo Marchiori M. (ed) (1998). QL'98 - The Query Languages Workshop, World Wide Web Consortium. See `http://www.w3.org/QL/QL98`

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

[Mengel, 1999b] Mengel, A., Dybkjaer, L., Garrido, J.M., Heid, U., Klein, M., Pirrelli, V., Poesio, M., Quazza, S., Schiffrin, A. & Soria, C. (1999). MATE Dialogue Annotation Guidelines, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart. See `http://www.ims.uni-stuttgart.de/projekte/mate/mdag`

[Michailovsky, 1999] Michailovsky, B., Lowe, John B. & Jacobson, M. (1999). Lacito Archivage Tool, Linguistic Data Archiving Project. See `http://lacito.vjf.cnrs.fr/ARCHIVAG/ENGLISH.htm`

[Murata, 1998] Murata, M. &Robie, J. (1998). Observations on Structured Query Languages., World Wide Web Consortium. See `http://www.w3.org/TandS/QL/QL98/pp/murata-san.html`.

[Sperberg-McQueen, 1994] Sperberg-McQueen, C. M. & Burnard L. (eds) (1994). *Guidelines for Electronic Text Encoding and Interchange*, Text Encoding Consortium, Oxford. See also `http://www.tei-c.org`.

[Thompson, 1999] Thompson. H.S., Beech, D., Maloney, M. & Mendelsohn, N. (1999). XML Schema Part 1: Structures. World Wide Web Consortium, Working Draft December 1999. See `http://www.w3.org/TR/xmlschema-1/`

[Vatton, 1999] Vatton, I., Guétari, R., Kahan, J. & Quint, V. (1999). Amaya – W3C's Editor/Browser, World Wide Web Consortium. See `http://www.w3.org/Amaya`

[Wood, 1998] Wood L. (ed) (1998). Document Object Model (DOM) Level 1 Specification Version 1.0, World Wide Web Consortium, December 1998. See `http://www.w3.org/TR/1998/REC-DOM-Level-1`.