The MATE workbench annotation tool, a technical description

Amy Isard[*], David McKelvie[*], Andreas Mengel[†], Morten Baun Møller[‡]

[*]HCRC Language Technology Group
Division of Informatics, University of Edinburgh
{amyi, dmck}@cogsci.ed.ac.uk

[†]Institut für Maschinelle Sprachverarbeitung
Universität Stuttgart
mengel@ims.uni-stuttgart.de

[‡]Natural Interactive Systems Laboratory, Odense University
baun@mip.sdu.dk

Abstract

The MATE workbench is a tool which aims to simplify the tasks of annotating, displaying and querying speech or text corpora. It is designed to help humans create language resources, and to make it easier for different groups to use one another's data, by providing one tool which can be used with many different annotation schemes. Any annotation scheme which can be converted to XML can be used with the workbench, and display formats optimised for particular annotation tasks are created using a transformation language similar to XSLT. The workbench is written entirely in Java, which means that it is platform-independent.

1. Introduction

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; Goldfarb and Prescod, 1998) and by allowing the corpus designer to write rule based transformations using a language very similar to Extensible Stylesheet Language Transformations (XSLT) (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. 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, 2000). 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.

The MATE workbench draws on several existing strands of work to provide a generic annotation tool which supports non-hierarchical markup (particularly necessary for speech due to overlapping annotation schemes and multiple overlapping speakers). The idea of using XML as a annotation standard for natural language processing goes back to the work of the Text Encoding Initiative (Ide and Véronis, 1995), and the idea of using a database as a central resource for natural language processing was developed by the GATE project (Cunningham et al., 1996). Data models related to XML and associated query languages were developed by a number of groups including the Lore project (Goldman et al., 1999) and SgmlQL (LeMaitre et al., 1996), and the idea of document transformation using structurally recursive rules comes originally from the work of the DSSSL (Adler, 1997) standard and XSLT working group. 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. The Amaya web editor (Vatton et al., 1999) takes a similar approach.

2. Workbench Architecture

The MATE workbench consists of the following major components, illustrated in figure 1:

- an internal database (see section 3.) 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;
- a query language and processor (see section 4.) which are used to select parts of this database for subsequent display or processing;
- a stylesheet language and processor (see section 5.1.) 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;
- a display processor (see section 5.2.) 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;

- a user interface (see section 6.) which handles file manipulation and tool invocation.

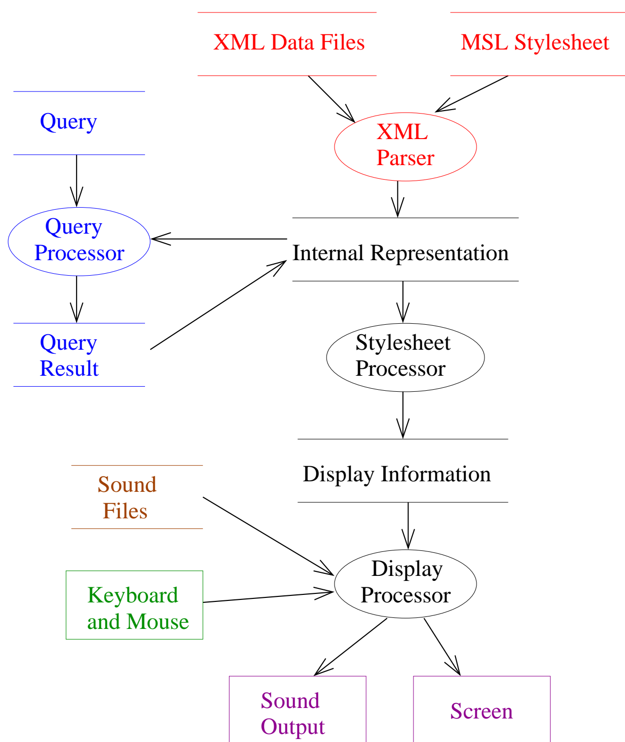

Figure 1: The MATE workbench architecture

3. XML and the MATE Internal Representation

3.1. Internal Database

When XML files are loaded into the workbench, the data are stored in an internal database. 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 an 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, for example 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 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 et al., 2000), Document Type Definitions (DTDs) are also represented as objects in the database. Some of 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.

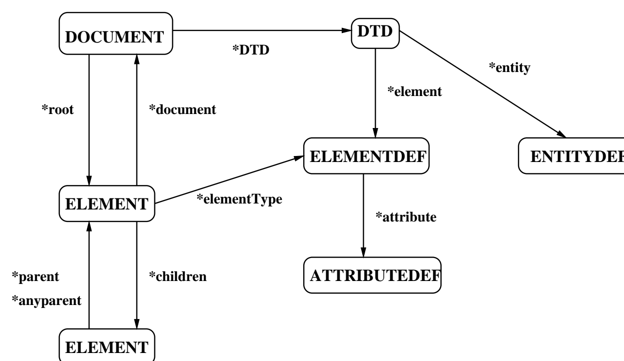

Figure 2: A partial schema for the MATE internal representation

3.2. Overlapping Hierarchies

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 overlap, for example 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) 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. We keep each level of annotation, and each data-stream (in the case of multi-speaker conversations for instance) separate, and link each further level of annotation to a common base-level. This base level would normally be the smallest unit on which all the other annotations depend. This may often be the word level, but could also be phonemes in the case of speech, higher level units such as sentences or paragraphs or indeed anything else as appropriate. The MATE workbench will therefore deal appropriately with any data which are marked up in this way.

4. Queries

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. it is possible to have 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 has 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 data (Abiteboul et al., 2000). We nonetheless chose to develop our own query language and processor (Q4M) because 1) when we started our work, there was no XML query module available in Java; 2) there is still no standard in this area;

(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 attribute value is ”adv” AND
(8):	`($w.who ~ "Peter") &&`	<word> who attribute value is ”Peter” AND
(9):	`($w @ $p) &&`	<pros> element occurs during the <word> AND
(10):	`($p.type ~ "H*")`	<pros> type value is “H*”

Figure 3: An example Q4M query.

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. 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 possible with all query languages, for example early versions of the semistructured data model. Thirdly, 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. 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, wild cards, 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 and combinations of simple expressions can be grouped together with parentheses. The query in figure 3 also demonstrates the use of time relations available in Q4M, e.g. '@' (temporal inclusion) and '][' (temporal contact). See (Mengel, 1999a; Heid and Mengel, 1999) for 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.

5. Stylesheets and Display

5.1. MATE 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 (Clark, 1999), 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), which uses the MATE query language, but is otherwise very 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. A fragment of a stylesheet is shown in figure 7.

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 users to write specific editors for particular annotation tasks. To provide this flexibility, firstly we assume that the visual appearance of a displayed document can be decomposed into display objects which form a hierarchical structure, for example XHTML or Java display objects, and 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.

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. 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. The project has developed annotation schemes for five sets of linguistic phenomena (Klein et al., 1998; Mengel, 1999b), and examples of markup using these schemes will be distributed with the workbench, along with stylesheets for their annotation and display. Users of the workbench are by no means limited to these schemes, however.

5.2. 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. These MATE display objects are used for creating displays for coding corpora or for showing query results. Each display object has a set of properties, which can be set, either directly in a Java program, or by running the Stylesheet Processor (with a MATE stylesheet and one or more XML files as input). The main

components used for building a display are panes (used to divide the display into two or more sections), vertical lists, horizontal lists, and text boxes (see figure 6 below for an example display).

5.3. Display Actions

In order to allow the user to interact with displays built from MATE display objects, we have added certain action properties to each object. These specify, for example, the behaviour which will occur if a user clicks on an object in the display, and when one of these display actions occurs, code in one of the attributes of the display object defines what is to happen. For example, if an XML file is associated with a speech file (through the use of start and end time attributes on word elements) then an action can be written which causes the relevant section of speech to be played when a word is clicked on. 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, and 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 yet entirely satisfied with this action syntax, and 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.

6. User Interface

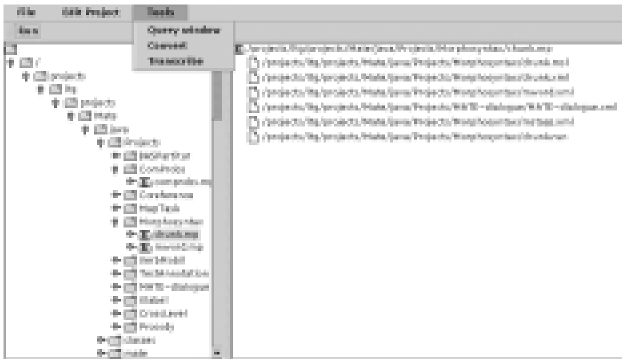

Figure 4: File Chooser Menu

When the workbench is started, a file chooser window (figure 4) appears, and the user can select a project, causing a list of the files referenced by the project file to appear in the right-hand window. The project can then be run to display a particular interface which consists of one or more windows, each contraining a speech player, or a number of panes of text. An example interface is shown in figure 6.

6.1. Tools

Various tools can be called from the file chooser window menus. The querying functions of the workbench 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 and 2) if a display has already been created, the files which make up the project used to create it 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 shown as a list, and if a display is already present, clicking on an element in the list repositions the main display on that element.

We currently have two available file format conversion tools. One converts from Entropics Xlabel format (Entropic, 1996) 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 converts from BAS Partitur format (Schiel et al., 1998) into an XML format.

6.1.1. Audio Player

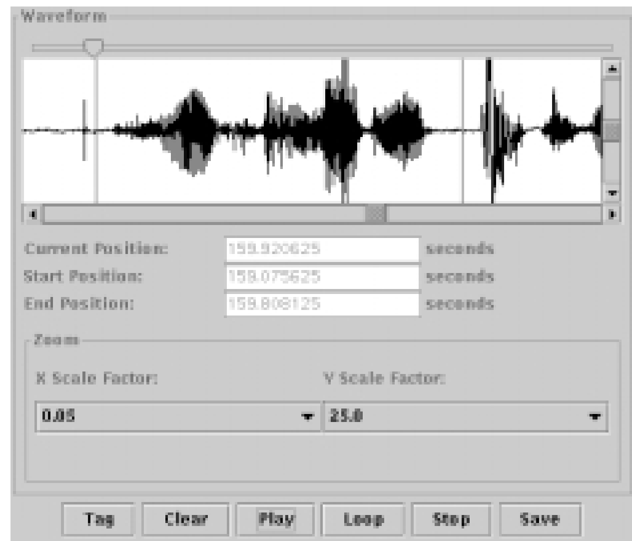

Figure 5: Audio Player Window

An audio player window (figure 5) 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 choose menu. This tool provides a scalable waveform display and standard playback functions. When a segment of speech has been selected, it is also possible to use a very simple transcription function, which produces an XML element with 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 workbench for large-scale transcription exercises as there are existing tools well suited to this purpose.

6.1.2. Coding Module Editor

If the user enters the workbench to carry out an annotation task, she/he first selects a coding module. A coding module prescribes what constitutes a coding, including the representation of markup and the relations to other codings. Users can produce new coding modules or view a pre-existing coding module along with its corresponding coding files using the MATE Coding Module Editor (CME), an intuitive user interface integrated into the MATE workbench.

Within the MATE workbench, the coding module and coding files are represented in XML providing an easy and precise parsing of the module and its corresponding coding files. However, using the CME to compose a coding module will only require a minimal knowledge of XML; the tool itself automatically generates the XML DTD from the element structure defined by the user. The markup declaration section of the coding module is represented as a tree, and the user adds elements, attributes, entities and comments to the tree to construct the markup declaration. For each node the name, type, etc. is specified. The tree can be parsed to create a coding module text document. The markup declaration node and its sub-nodes contain information that can be used to create a DTD which is used internally in the Mate workbench.

6.2. Example Display

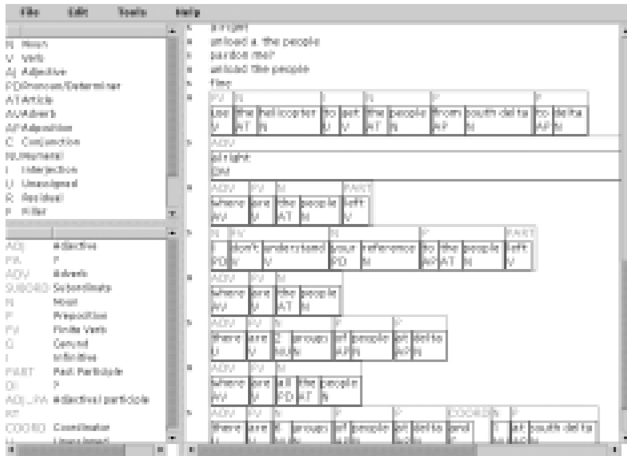

Figure 6: Morphosyntax Display

Figure 6 shows a display of morphosyntactic annotation. The display is built of three panes, each of which consists of a vertical list made up of horizontal lists which in turn are made up of text boxes. Figure 7 shows a fragment of the stylesheet used to create the display; this template matches all word elements which are the last child of a chunk element, and creates the top level of boxes in the example. Further templates match words in other contexts, to build up the whole picture.

7. Evaluation

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.

Information on how to obtain a copy of the MATE workbench can be found at http://mate.nis.sdu.dk

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.

```
<msl:template match="($w word)($ch ch);($ch 2^m $w)">
 <MVerticalList Alignment="Left"
    FrameVisible="True" FrameColor="Magenta">
  <msl:apply-templates
     select="($ch ch);($ch 2^ $this)"/>
  <MHorizontalList Alignment="Left">
   <msl:for-each
      select="($w word)($ch ch)($mw mw);($ch 2^m $this)
      and ($ch 2^n $w) and ($mw 1^m $w)">
    <MVerticalList Alignment="Left"
       FrameVisible="True" FrameColor="Blue">
     <MHorizontalList Alignment="Left">
      <msl:for-each
         select="($w word)($mw mw);($mw 1^m $this)
         and ($mw 1^ $w)" cache="no">
       <MTextEditor FontSize="12">
        <msl:apply-templates/>
        <msl:text> </msl:text>
       </MTextEditor>
      </msl:for-each>
     </MHorizontalList>
     <MHorizontalList Alignment="Left">
      <msl:apply-templates
         select="($mw mw);($mw 1^m $this)" cache="no"/>
     </MHorizontalList>
    </MVerticalList>
   </msl:for-each>
  </MHorizontalList>
 </MVerticalList>
</msl:template>
```

Figure 7: Fragment of an MSL stylesheet

9. References

Abiteboul, Serge, Peter Buneman, and Dan Suciu, 2000. *Data on the Web: from relations to semistructured data and XML*. San Francisco, California: Morgan Kaufmann.

Adler, Sharon C., 1997. The “ABCs” of DSSSL. *Structured Information/Standards for Document Architectures*, 48(7):597–602. Journal of the American Society for Information Science, Special Issue.

Bird, Steven and Mark Liberman, 2000. Linguistic annotation resources. Linguistic Data Consortium. www.ldc.upenn.edu/annotation.

Bray, Tim, Jean Paoli, and C. M. Sperberg-McQueen, 1998. Extensible Markup Language. World Wide Web Consortium. www.w3.org/TR/REC-xml.

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

Cunningham, Hamish, Yorick Wilks, and Robert J. Gaizauskas, 1996. GATE – a general architecture for text engineering. In *Proceedings of the 16th International Conference on Computational Linguistics*. Copenhagen, Denmark. See also www.dcs.shef.ac.uk/research/groups/nlp/gate.

DeRose, Steve, Ron Daniel Jr., and Eve Maler, 1999. XML Pointer Language (XPointer). World Wide Web Consortium. www.w3.org/TR/xptr.

DeRose, Steve, Eve Maler, David Orchard, and Ben Trafford, 2000. XML Linking Language (XLink). World Wide Web Consortium. www.w3.org/TR/xlink.

Entropic, 1996. *Waves+ Manual*. Entropic Research Laboratory Inc. www.entropic.com.

Goldfarb, Charles F. and Paul Prescod, 1998. *The XML Handbook*. Upper Saddle River, NJ: Prentice Hall.

Goldman, Roy, Jason McHugh, and Jennifer Widom, 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.

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

Ide, Nancy and Jean Véronis (eds.), 1995. *The Text Encoding Initiative: Background and Context*. Dordrecht: Kluwer. See also www.tei-c.org.

Isard, Amy, David McKelvie, and Henry S. Thompson, 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.

Klein, Marion, Niels Ole Bernsen, Sarah Davies, Laila Dybkjaer, Juanma Garrido, Henrik Kasch, Andreas Mengel, Vito Pirelli, Massimo Poesio, Silvia Quazza, and Claudia Soria, 1998. MATE deliverable 1.1: Supported coding schemes. mate.nis.sdu.dk/about/deliverables.html.

LeMaitre, Jacques, Elisabeth Muriasco, and Monique Rolbert, 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, Massimo, 1998. QL'98 - the Query Languages Workshop. World Wide Web Consortium. www.w3.org/TandS/QL/QL98.

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

Mengel, Andreas, 1999b. MATE deliverable 2.1: DTDs and notes on schemas. mate.nis.sdu.dk/about/deliverables.html.

Murata, Makoto and Jonathan Robie, 1998. Observations on structured query languages. In Massimo Marchiori (ed.), *Proceedings of QL'89 - The Query Languages Workshop*. Boston, Massachussets. www.w3.org/TandS/QL/QL98/pp/murata-san.html.

Schiel, Florian, Susanne Burger, Anja Geumann, and Karl Weilhammer, 1998. The partitur format at bas. In *Proceedings of the First International Conference on Language Resources and Evaluation*. Granada, Spain.

Vatton, Irène, Ramzi Guétari, José Kahan, and Vincent Quint, 1999. Amaya – W3C's editor/browser. World Wide Web Consortium. www.w3.org/Amaya.

Wood, Lauren, Arnaud Le Hors, Vidur Apparao, Laurence Cable, Mike Champion, Mark Davis, Joe Kesselman, Philippe Le Hégaret, Tom Pixley, Jonathan Robie, Peter Sharpe, and Chris Wilson, 2000. Document Object Model (DOM) level 2 specification. World Wide Web Consortium. www.w3.org/TR/DOM-Level-2.