

## MUP: The UIC Standoff Markup Tool

Michael Glass and Barbara Di Eugenio

CS. Dept. M/C 152, University of Illinois at Chicago

851. S. Morgan

Chicago, IL, 60607-7053

{mglass|bdieugen}@cs.uic.edu

### Abstract

Recently developed markup tools for dialogue work are quite sophisticated and require considerable knowledge and overhead, but older tools do not support XML standoff markup, the current annotation style of choice. For the DIAG-NLP project we have created a “lightweight” but modern markup tool that can be configured and used by the working NLP researcher.

### Introduction

Speech and text corpora augmented with linguistic annotations have become essential to everyday NLP. In the realm of discourse-related annotation, which we are interested in, linguistic annotation is still mostly a manual effort. Thus, the availability of coding tools that facilitate a human coder’s task has become paramount. In this paper we present MUP, a coding tool for standoff markup which is sophisticated enough to allow for a variety of different markings to be applied, but which is also simple enough to use that it does not require a sizable set up effort.

Other coding tools have been developed, and some of them do in fact target discourse phenomena. Tools specifically developed to code for discourse phenomena include Nb (Flammia and Zue, 1995), DAT (Allen and Core, 1997), MATE (McKelvie et al., 2001), and the Alembic Workbench (Day et al., 1997). MUP differs from all of them because it is standoff (contrary to Nb and DAT), allows tagging of discontinuous constituents (contrary to Nb), and is simple to set up and use (contrary to MATE).

We developed MUP within the DIAG-NLP project (Di Eugenio et al., 2002), which is grafting an NLG component onto a tutorial program written in the VIVIDS (Munro, 1994) and DIAG (Towne, 1997) ITS authoring environment. MUP is targeted to written or transcribed text. Phenomena such as intonation contours and overlapping speech have no opportunity to occur in our transcripts. Thus MUP lacks features that spoken-language phenomena require of annotation tools, e.g. layers of annotation to repair disfluencies, the representation of simultaneous speakers, and interfaces to speech tools.

### Requirements and Alternatives

Our fundamental requirements for a markup tool are that it 1) use standoff markup, 2) represent source documents, annotations, and control files in simple XML, 3) have a simple graphical annotation interface, 4) provide control over element names, attribute names, and attribute values, enforcing consistency in the final markup, and 5) can be configured and employed by everyday computational linguists without much effort or training.

In standoff markup (Thompson and McKelvie, 1997) the source text is inviolate and the annotations are kept physically separate, usually in other files. Annotatable items in the source text contain labels, while the physically separate annotations refer to these labels. Since annotations are themselves labeled, complex structures of linked annotated constituents pointing to each other are representable.

Thompson and McKelvie list three advantages to standoff markup: 1) the source document might be read-only or unwieldy, 2) the annotations can deviate from the strictly tree-structured hierarchies that in-line XML demands, 3) annotation files can be

distributed without distributing the source text. We note a few more advantages of the standoff style: 4) discontinuous segments of text can be combined in a single annotation, 5) independent parallel coders produce independent parallel annotation files, aiding the determination of inter-coder reliability, 6) different annotation files can contain different layers of information, 7) when the source text is regenerated from primary sources (for example, to incorporate more information) existing annotations are preserved.

Several years before Thompson and McKelvie, the Tipster project evolved a similar standoff architecture for similar reasons (Grishman, 1995). Two notable differences between Tipster and our own architecture are that Tipster annotations refer to absolute byte numbers within an unlabeled source document file, and the Tipster architecture does not use XML or SGML but instead supports its own class library and internal representation. Other markup projects, for example the Alembic Workbench, have taken their cue from Tipster and implemented the same standoff idea.

We specified XML for an annotation language because it is a lingua franca: the vocabulary is quite commonly known, there is a host of XML processing software, people can inspect it, and XML provides a rich ability to add attributes to elements.

The ATLAS.ti (Muhr, 2002) and the NUD\*IST annotation packages (QSR Corp., 2002) have both been marketed for many years to researchers in the “soft” sciences for computer-assisted qualitative analysis of texts. Their emphasis is on visually illustrating the various codes attached to parts of the document so the researcher can observe patterns. Fairly complicated relationships between the annotation tags can be created and visualized. An impressive characteristic of these packages is that ordinary researchers in non-technical fields can create their own tags and commence annotating. However the annotations in these packages point to sections of the plain-text source document by absolute byte number. Thus the annotations are not readily available for inspection or machine processing outside of the programs’ own interfaces and existing XML-tagged data cannot readily become a source document for further markup. These packages are not useful for the analysis usually needed by NLP projects. It is

```
<xscript id="t12">
...
<tutor-resp id="t12_turn_3">
<w id="t12_t19">to</w>
<w id="t12_t20">see</w>
<w id="t12_t21">if</w>
<w id="t12_t22">the</w>
<w id="t12_t23">oil</w>
...
```

Figure 1: Source Document: “to see if the oil...”

interesting to note that the most recent versions of ATLAS.ti and NUD\*IST have been adding the ability to import and export structured documents and annotations, in XML and Rich Text Format respectively.

At another extreme are markup tools like DAT, an annotator for DAMSL (Allen and Core, 1997), a dense multi-layered annotation scheme rich with attributes. DAT is extremely convenient for the coder, but it seems to require expert reprogramming when DAMSL’s tag set changes.

### A Taste of MUP

Running MUP requires a source document, a DTD-like document describing a tag set, a style file controlling how source text and annotations are displayed to the user, and optionally an existing annotation file. The coder can then mark up the document and save the results.

**Source Document** Figure 1 shows an extract from a DIAG-NLP project source document. These source document elements have special meaning: word elements in line-wrapped text are tagged `<word>` or `<w>`, and formatted lines of text are tagged with `<line>` to be displayed without wrapping. These elements must be labeled with XML ID attributes to be the target of annotations. Other XML elements may appear in the source document as desired. All source document elements can be optionally revealed or hidden for the coder, styled according to the style file.

**Tag Descriptions** Each tag is an empty XML element, described in the tags description file by an `<!ATTLIST>` declaration. We omit `<!ELEMENT>` declarations as superfluous, so this file is not fully a DTD. The pop-up dialogue the coder uses for entering and editing attributes is driven by the

<!ATTLIST> description. Figure 2 illustrates the description of a tag named `indicator`. The `id`, `idrefs`, and `comment` attributes are standard for all MUP markup tags. This description is interpreted, in part, as follows:

- `idrefs` will contain the IDs of ranges of target elements for this annotation, selected by the coder by painting the source text with a cursor.
- The `comment` attribute, being CDATA, contains arbitrary text typed by the coder.
- The `directness`, `senseref` and `indicator_name` attributes, being enumerated lists, will present a drop-down list of values to the coder. Notice the `indicator_name` is specified by entity substitution for convenience.

**Snapshot of MUP at Work** Figure 3 shows a snapshot of MUP at work. A control window has a list of available markup tags plus it shows which source document elements are displayed or hidden, while the source document text is in a separate window. The style file controls the display of the source document by selecting which elements and attributes to show/hide, picking colors for highlighting, and inserting some bracketing text before and after. In the snapshot, we have elected to display the data from the `<date>` tag, the `<consult>` element with its attributes but not the data, and the `<tutor-resp>` element and data with a separator after it, while suppressing all other elements. We have shown the text “the blue box that lets you see if the oil is flowing” being annotated as an `indicator` via a pop-up dialogue.

## Discussion

We believe a simplified easy-to-configure and run tool will have wider applicability beyond our own project. Other projects that manually code large quantities of typed text, e.g. the RST dialogue markup project (Marcu et al., 1999), have found it desirable to create their own markup tools. The CIRCSIM-Tutor project, with well over a hundred transcripts of typed dialogue averaging an hour each, has been coding in SGML (Freedman et al., 1998; Kim, 1999) with general-purpose text editors.

The MATE workbench (McKelvie et al., 2001) is a full-featured dialogue markup tool, however we found it to be complex and difficult to use. We saw an opportunity to borrow some of the ideas from MATE and realize them with a simpler annotation tool. MATE envisions three levels of user: coders, researchers for whom the coding task is performed and who need to view and manipulate the results, and experts who are able to configure the software (Carletta and Isard, 1999). It is this last group that can perform the manipulations necessary for adding new tags to the tag set and controlling how they are displayed. MATE permits programmatic control over the coding interface by means of an XSL style sheet customized for a particular application. It is possible to split windows, intercept cursor operations, provide linking operations between text in different windows, and so on. This kind of flexibility is useful in annotated speech, for example in separately displaying and linking two speech streams and having several related windows update simultaneously in response to coder actions. In our experience the MATE style sheets were quite difficult to write and debug, and for our application we did not need the flexibility, so we dispensed with them and created our own, simpler, mechanism to control the display of text. One consequence of the lessened flexibility in MUP is that it presents a consistent coding interface using familiar single-dialog GUI conventions.

Brooks (1975) estimates that the difference between a working program and a usable system with ancillary utilities, shell scripts, etc. is three times the original effort, and producing a distributable product requires another factor of three effort. The core MUP program works well enough that we have been using it for several months. Our highest priority next enhancement is to add a utility for inter-rater comparison, featuring some control over how parallel annotations are compared (e.g., by selecting which of the element’s attributes must match), and automatically computing  $\kappa$  statistics. MUP runs on Solaris and Linux. We will make it available to researchers as it matures.

## Acknowledgments

This work is supported by grant N00014-00-1-0640 from the ONR Cognitive, Neural and Biomolecular S&T Division and

```

<!ENTITY % indlist "( current-temp-gauge | sight-hole | water_temp_gauge ...)" >
<!ATTLIST indicator
  id ID #required
  idrefs IDREFS #required
  comment CDATA #IMPLIED
  indicator_name %indlist; 'unspecified'
  directness ( explicit | implicit | summary | unspecified ) 'unspecified'
  senseref ( sense | reference | unspecified ) 'unspecified' >

```

Figure 2: Description of indicator tag

Research Infrastructure grant EIA-9802090 from NSF. Thanks also to Maarika Traat and Heena Raval, who have been most helpful in the DIAG-NLP markup effort.

## References

- James Allen and Mark Core. 1997. Draft of DAMSL: Dialog Act Markup in Several Layers. <http://www.cs.rochester.edu/research/cisd/resources/damsl/>.
- Frederick P. Brooks. 1975. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, Reading, MA.
- Jean Carletta and Amy Isard. 1999. The MATE annotation workbench: User requirements. In Marilyn Walker, editor, *Towards Standards and Tools for Discourse Tagging: Proceedings of the Workshop, College Park MD*, pages 11–17, New Brunswick, NJ. Association for Computational Linguistics.
- David Day, John Aberdeen, Lynette Hirschmann, Robyn Kozierok, Patricia Robinson, and Marc Vilain. 1997. Mixed-initiative development of language processing systems. In *Fifth Conference on Applied Natural Language Processing ANLP-97*, pages 348–355, New Brunswick, NJ. Association for Computational Linguistics.
- Barbara Di Eugenio, Michael Glass, and Michael J. Troilo. 2002. The DIAG experiments: Natural language generation for intelligent tutoring systems. In *Second International Natural Language Generation Conference INLG '02, Harriman, NY*. To appear.
- Giovanni Flammia and Victor Zue. 1995. Empirical evaluation of human performance and agreement in parsing discourse constituents in spoken dialogue. In *Proc. Eurospeech-95, Fourth European Conference on Speech Communication and Technology*, pages 1965–1968.
- Reva Freedman, Yujian Zhou, Jung Hee Kim, Michael Glass, and Martha W. Evens. 1998. SGML-based markup as a step toward improving knowledge acquisition for text generation. In *AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 114–117.
- Ralph Grishman. 1995. Tipster phase II architecture design document (Tinman architecture). Technical report, New York University. <http://cs.nyu.edu/pub/nlp/tipster/152.ps>.
- Jung Hee Kim. 1999. A manual for SGML markup of tutoring transcripts. Technical report, CIRCSIM-Tutor Project, Illinois Institute of Technology. <http://www.cs.iit.edu/~circsim/>.
- Daniel Marcu, Estibaliz Amorrortu, and Magdalena Romera. 1999. Experiments in constructing a corpus of discourse trees. In Marilyn Walker, editor, *Towards Standards and Tools for Discourse Tagging: Proceedings of the Workshop, College Park MD*, pages 48–57, New Brunswick, NJ. Association for Computational Linguistics.
- Dave McKelvie, Amy Isard, Andreas Mengel, Morten Braun Møller, Michael Grosse, and Marion Klein. 2001. The MATE workbench – an annotation tool for XML coded speech corpora. *Speech Communication*, 33(1–2):97–112. <http://www.cogsci.ed.ac.uk/~dmck/Papers/speechcomm00.ps>.
- Thomas Muhr. 2002. ATLAS.ti home page. <http://www.atlasti.de/atlasneu.html>.
- Allen Munro. 1994. Authoring interactive graphical models. In T. de Jong, D. M. Towne, and H. Spada, editors, *The Use of Computer Models for Explication, Analysis and Experiential Learning*. Springer Verlag.
- QSR Corp. 2002. NUD\*IST home page. <http://www.qsr.com.au/>.
- Henry Thompson and David McKelvie. 1997. Hyperlink semantics for standoff markup of read-only documents. In *SGML Europe 97, Barcelona*. <http://www.infloom.com/gcaconfs/WEB/TOC/barcelona97toc.HTM>.
- Douglas Towne. 1997. Approximate reasoning techniques for intelligent diagnostic instruction. *International Journal of AI in Education*, 8:262–283.

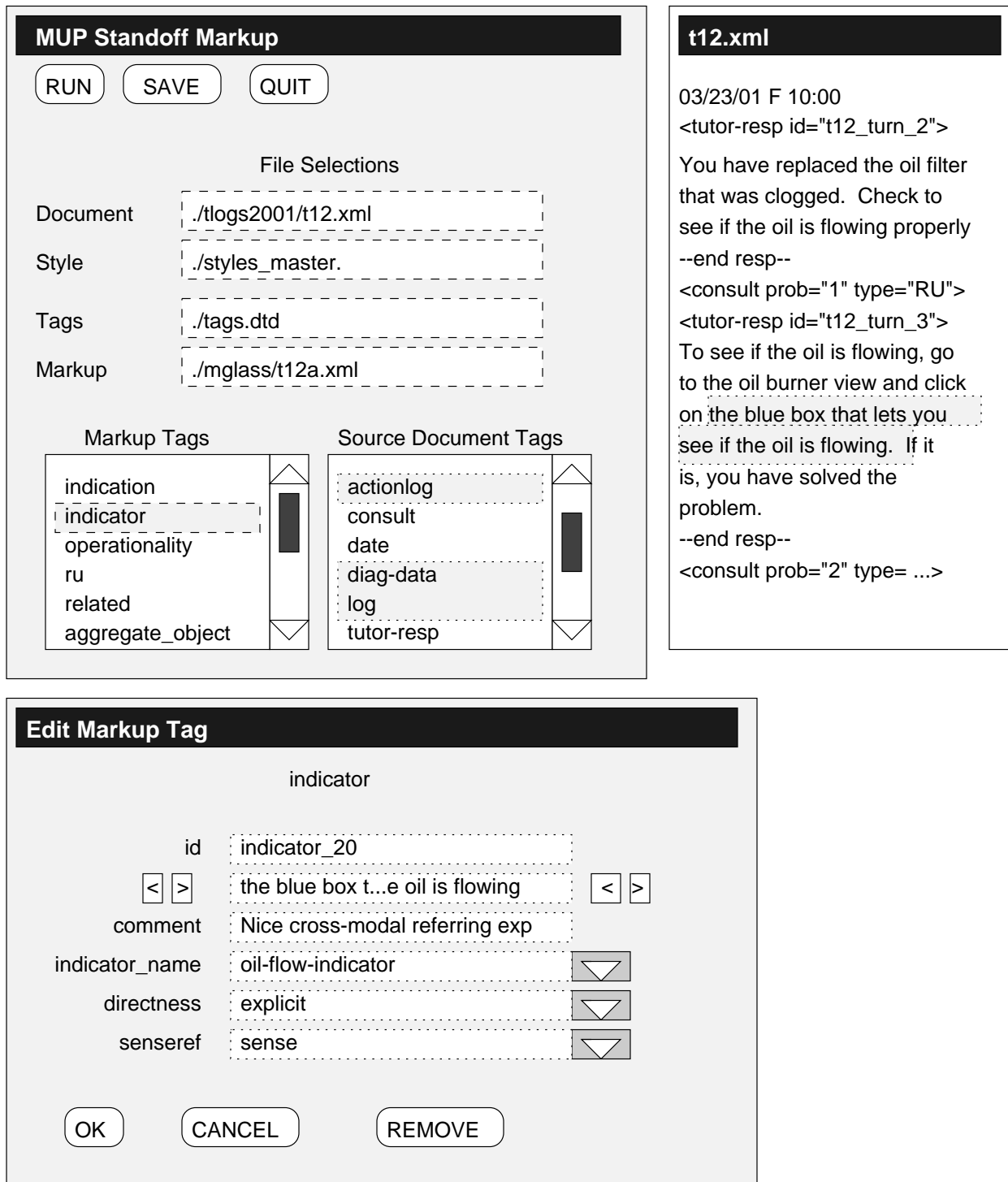


Figure 3: MUP in Action: Control Panel, Text Window, and Edit Tag Window