ELECTRONIC PUBLISHING, VOL. 6(4), 349–360 (DECEMBER 1993)

# A constraint-based editor for linguistic scholars

ROBERT A. MORRIS
*Department of Mathematics and Computer Science*
*University of Massachusetts*
*Boston, MA 02125, USA*

*e-mail:* `ram@cs.umb.edu`

EDWARD M. BLACHMAN
*Member of the Technical Staff*
*Interleaf, Inc.*
*Boston, MA, USA*

CHARLES MEYER
*Department of English*
*University of Massachusetts*
*Boston, MA 02125, USA*

**SUMMARY**
**A constraint-based interactive structure editor for use by linguists is described. Multiple, inter-related constraint sets are supported. A novel search mechanism is introduced which modifies itself locally dependent on document structure as the search progresses.**

KEY WORDS Constraint-based Corpus linguistics Markup SGML ICE Text Encoding Initiative

## 1 INTRODUCTION

Corpus linguistics is a branch of linguistics in which the scholars analyse large collections of electronic documents. For English there are many such corpora. The earliest and best known is the Brown Corpus [1]. The Linguistic Data Consortium at the University of Pennsylvania is collecting several hundred million words of English. The Text Encoding Initiative (TEI) [2] is specifying SGML Document Type Definition standards for corpus linguistics, which have already been put to use in the 100 million word British National Corpus (BNC) [3]. The International Corpus of English (ICE) [4], of which one of us (Meyer) leads the American effort, is collecting 1 million words of written and spoken English from each of 15 countries.

To be useful for electronic analysis, documents once transcribed into electronic form must have some conventional markup inserted to mark the boundaries of whatever linguistic constructs are to be studied. Some of this markup, such as that implied by the natural language parsing, can be inserted by parsing programs with, perhaps, a small amount of human intervention to disambiguate the parse (cf. [5]). When the transcription is of spoken language, the state of natural language parsing is inadequate to this task, and even for written text some things do not yield to machine marking. For example, in order to support correct analyses, mis-spellings may be corrected by a (human) corpus editor, and the corpus must retain both the original and the ‘normalization’, both suitably marked.

Therefore, perhaps after initial machine markup a document must have markup added by a human, typically with some linguistic training. This is a time consuming task, prone to logical errors if done without adequate software. Our program supports the addition of ICE markup in ways that make impossible the addition or deletion of markup in violation of constraints defined by the markup scheme, while at the same time permitting the full editing of the text itself. The constraints are deduced from a hierarchy we have placed on the ICE markup (none is officially specified), but are easily changed.

CCC 0894–3982/93/040349–12
Received 15 August 1993
Revised 1 December 1993

Editor constraints are familiar in the structured editor domain. For example, chapters must not appear inside sections, paragraphs inside other paragraphs, etc. There are many ways to specify the constraint graphs, including standardized schemes such as SGML, and there are many familiar editors which are cognizant of those schemes. The most famous is the Cornell Program Synthesizer. Many structured document preparation systems also enforce constraints on their structural elements. We wish to describe the experiences we had in building such an editor for linguists, and to describe what we believe to be some novel features.

One particular difficulty, whose solution we describe below, is that ICE markup supports description of speech overlap in spoken transcriptions. The structure of this speech synchrony information is more limited than in some schemes (see [6] for a survey), but the result is that there are two overlapping but related constraint sets — one for the synchrony and one for the linguistic structure.

## 2 THE CONSTRAINT REPRESENTATION

Our constraint graph is implemented as a set of named categories of tags. In the simplest case, each category is associated with a list of permitted subcategories. For example, the category named *textblock* (a document structuring construction) has members *footnote*, *marginalia*, *heading*, *paragraph* and *discourse-text* and child categories *utterance*, *quotation* and *pause*. The category *quotation* has children *extra-corpus-text*, *textblock*, *quotation* and *speech* in the structure graph. The directed subgraph rooted at *textblock* is shown in Figure 1, which implies we could have a document structure like:

```
    paragraph
        quote
            paragraph
```

or

```
    quote
        paragraph
            quote
```

but not

```
    paragraph
        paragraph
            quote
```

A graph like that of Figure 1 or an equivalent finite state machine is frequently used to parse a document for structural correctness. However, in traditional document processing, a single parse tree is insufficient if one wishes to treat the document layout in the same terms as its structure. A common approach for WYSIWYG editors is to deduce layout from property lists and structure by a hierarchy of special objects throughout the document. Many commercial systems, e.g., Interleaf, FrameMaker, and some desktop publishing systems make such a distinction, as do such research systems as Quill [7] and Grif [8].

When layout and structure are to be treated similarly, the difficulty is that hierarchy relations might overlap one another. For example, pages contain text and paragraphs contain the same text, but in general, pages can't contain paragraphs and, conversely, paragraphs can't contain pages. Our solution to problems of this sort is to introduce *multiple* rooted

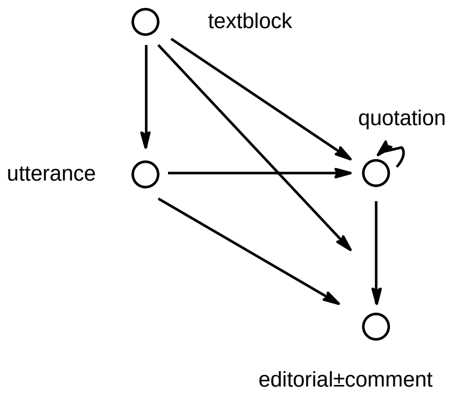

*Figure 1. Directed subgraph rooted at the textblock category*

subgraphs for constraint sets, with each edge coloured to indicate to which constraint set the subcategory relation belongs.

In ICE markup, the transcription of dialogues containing overlapping speech creates a requirement for two such colors, one denoting the *structure* graph and one the *synchrony* graph. ICE introduces two kinds of tags to deal with overlapping speech: one which denotes the start and end of a zone in which overlap occurs, and one which denotes the start and of end of a particular speaker's participation in a particular episode of overlap. This latter causes the turn boundary tag, which is used even in non-overlapping speech to indicate the onset of each contribution of each speaker, to take on new meaning: namely a speaker's contribution to an episode of overlap may not cross a turn boundary. Correspondingly, note that transcriptions with no overlaps do not require two graphs. Such a transcription, when fully marked up, can have its structure described with a single parse tree.

In ICE markup, a small overlap might be marked like this:

```
<$A> I <{\_> <[_> thought you might <[/> not

<$B> <[_> you thought I might what <[/> <{/>

<$A> Oh, never mind.
```

In this fragment, <$A> and <$B> identify speakers A and B, and the tags <{_> and <{/> indicate the start and end of the collection of overlapping strings. The tags <[_> and <[/> indicate the start and end of particular overlapping strings. For more detail on complexities not present in such a simple overlap, see our paper [6], For simplicity, we have omitted an ICE tag required at each speaker turn boundary which includes a serial number to facilitate text searches. Also, see below for the visually simpler way we present this overlap to users of our system.

Aside from its root, only three categories lie in the synchrony graph, each containing a single member. The first of these, *overlapping-set*, contains only the tag type also called *overlapping-set*. Such tags mark the boundaries of a conversation with overlap. The *speech* category contains the tag type called *speaker-id*. *speech* nodes lie in both the structure

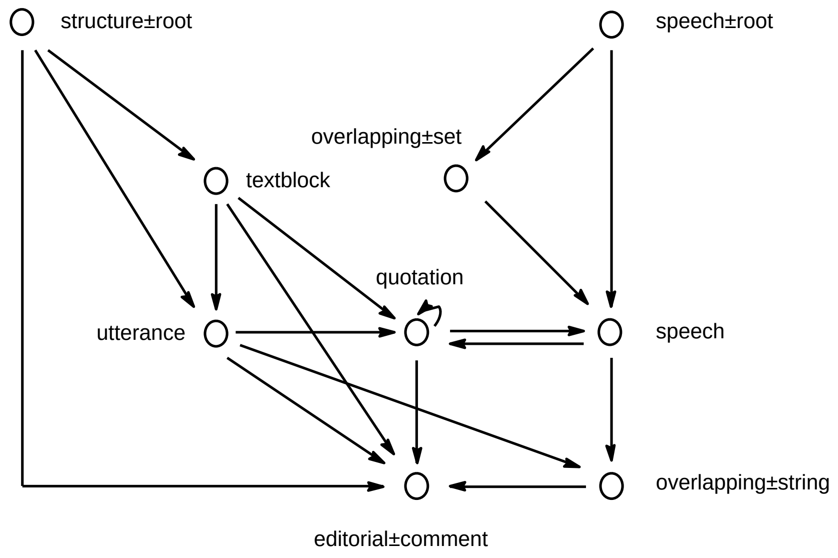

*Figure 2. A doubly rooted subgraph of the full category graph*

graph and the synchrony graph (when it is present). Finally, the *overlapping-string* category contains only the tag type marking the boundaries of particular overlapping strings. It too can lie in the structure graph, but in this graph its only child is the somewhat trivial category *editorial-comments*. Although ICE distinguishes several types of editorial comments, they all reflect the addition of text to that under analysis. They can appear virtually anywhere in a document. See Figure 2 for a fragment of the category graph, which, however, contains the entirety of the synchrony ('red' or dotted line) graph and its connections to the structure ('black' or solid line) graph.

Categorizing tags in this fashion also has benefit for the user, who can internalize a smaller set of objects than the entire ICE tag set. To enforce the hierarchy, the user is never presented with a menu containing anything other than tags which meet the hierarchy constraints. We accomplish this with the algorithm below. An *element* is a string that contains at least a start-tag which identifies the element. Where the constraints allow the element to have content, the start-tag can be followed in the string by a mixture of text and other elements — which other elements can occur is dictated by the constraints — followed finally by an ending tag. Some tags are self-closing, in which case the inner content is empty. In SGML terms, to say that the inner content meets the constraints is to say that it satisfies the content model of the element.

## 2.1 Algorithm for tag menu

Define a *region* as a substring of the tagged text. Regions are denoted by a left edge and a right edge. Any point in the text is in fact also a 0-length region, denoted by left and right edges that match that point.

1. In each graph, the *parent* of a region is the first start-tag to the left of the region whose end-tag is to the right of the region, provided that the category of the start-tag has subcategories in the graph of interest. If there is no such tag, the parent is nil.

2. For each graph, consider for creation the subcategories of the category of the parent in that graph. If the parent is nil, use the subcategories of the graph's root. If the region is non-nil, further winnow the set of categories by using only those that can validly parent all top-level elements in the region.
3. For each category offered, generate a pullaside submenu listing all members of that category.
   In response to the choice made, the software drops a start-tag at the beginning of the region and, unless the element can contain no content, a corresponding end-tag at the end of the region. Where necessary, elements within the region are re-parented to the new element.

In practice, a user does not tag an empty document whose text gets entered interactively, but rather an unmarked text which has been produced by optical scanning, by transcription, or obtained electronically. In this case, the user selects a string for tagging (our selection mechanism forces this to contain only text and complete elements even if the user attempts to select a partial element) and the above algorithm is applied.

## 3 REMARKS ON IMPLEMENTATION

We built our editor on top of Interleaf Release 5.3 software. This is a commercial, extensible, object-oriented compound document editing system based on Lisp. The extension facilities have been described elsewhere [9]. The user inserts tags which are chosen from a hierarchical popup menu (an extension of the standard one), which appears when the appropriate mouse button is pressed. At each point in the document, only choices consistent with the constraints and the current markup are offered, using the algorithm described above. Taggers begin by initiating an Interleaf session. This gives them a desktop-metaphorical view of their filesystem. They select a corpus text and open it; this gives them a view of their text as a normal Interleaf document. The behavior of the editor is, in most respects, identical to that of the standard Interleaf software; however, in areas where a tagger would expect help (e.g., selection behavior and creation menus), the editor has been enhanced along the lines described above. Eventually the tagging is finished, and the tagger saves the document, at which point a new version of the corpus text is created, tagged according to the tagger's instructions.

One appeal for us of Interleaf software as a foundation is its extensibility. Document objects, the objects which give them structure and the *mechanism* for dealing with content and structure objects are available to the programmer in an object system with extensive subclassing facilities. The starting point for altering or extending behaviors is the document's class. Each document has a default class, which is used unless the document instructs otherwise when the system first notices it (typically at startup). We created a new document subclass, which gave us a hook on which to hang the other changes we made.

Editing interfaces are controlled by *editor objects*, which manage various parts of the user interaction, including menus and selection behavior. Our documents use specialized subclasses of the generic *document*-, *text*- and *table-editor* classes, as each of those controls a domain in which we implement changes.

To represent tags we use Interleaf's 'inline components'. These are named paragraph-like objects which can appear in arbitrary positions in text and can have in them most kinds of Interleaf document objects, including text, graphics, or other inline components.

Within those components, we add a mnemonic representation of the tag as plain text, so that if the tags are (optionally) displayed, the user can understand the tagged structure. However, these tags need special behavior. For example, the mnemonic text in them must be uneditable, and their cut behavior is constrained. We subclass the native inline component class to implement these behaviors. In order to support real-time computation of the context sensitive menus, the hierarchy of tags is represented by Lisp data on the tags.

Finally, the constraint graphs are stored as a list associating category names and structures. Each category structure contains its list(s) of subcategory names, and a list of its members; each of which is in fact a structure describing the element in more detail.

## 4 PROBLEMS ARISING FROM THE UNDERLYING SYSTEM

Originally based on the MIT Etude project [10], and first offered for sale in 1984, Interleaf software was one of the earliest commercial WYSIWYG compound document systems. It was externally object-oriented from the beginning, with a rigorous noun-verb model for its user interface. However, it was not made fully extensible until the current major release (5.x), first available in 1990.

Because the object programming subsystem was a late addition to a mature body of code, it has quirks that betray the interactive assumptions underlying the substrate. Consider searching, for example. In an interactive editor, it makes sense that each time a search is successful, the user’s view of the document would be repositioned to show the object that was found. It also makes sense in this context that the user’s view would give feedback on the state of the document at the given point (cursor location, font data, and so on). For our purposes, however, searching was just a way to build up information about the document — something that needed to be done quickly and *without* affecting the user. When we attempted to use the built-in search code, this clash of requirements made our initial efforts quite . . . flashy. The builders were not unaware of the need to suppress these display effects. However, the mechanisms they added to suppress display are neither complete nor uniform across subsystems. They ultimately proved insufficient to suppress distracting screen pyrotechnics in our case, so we rewrote the search code from scratch, abandoning the native search primitive. The net result is no faster than the original — but because it involves no screen redisplay, it’s no slower either, and it eliminated the distraction of the earlier approach.

In other areas, the programming subsystem is incomplete. For instance, it does not include a general persistent Lisp handle on document objects. While there is a cross-referencing mechanism in the software, it is limited to page and autonumber references and is not accessible for general objects. Therefore, we subclassed the standard document class to provide a special open method which reconstructs the tag hierarchy transparently to the user. As a Lisp object, every tag can be referenced and in Lisp every object has a print representation. In the case of components, this is essentially a string representation of its memory address. We store this name on the tag object so that it can be found at document open time. (Searching the document for inline components is an inexpensive operation.) Using the names, we recreate the links describing the hierarchy. At the same time, we *rename* the object to correspond to its current internal name. This is done to prevent a name collision should a new tag be created whose internal name happens to be the same as that which the given tag had in a previous open incarnation of the document.

Cutting and pasting reveal another area of incompleteness. It is not possible to access or examine document objects on the Interleaf clipboard, but we need access to those objects, because pasting an element must be permitted only if that element meets the hierarchy constraints at the point of paste. Therefore, we introduced a ‘pseudo-clipboard’ (pcb), a more-or-less ordinary Interleaf document in a well-known location, and provided our editor objects with special cut and paste methods which put the cut objects onto the pcb instead of the system clipboard. Except during debugging, the pcb is normally an invisible document, which aside from display and input behaves in all respects like any other document. (However, when made visible for debugging purposes, cutting from an ICE document has the startling effect of making the cut text appear in the pcb with no further user interaction.) Because the pcb, like all open Interleaf documents, has its own internal name space, we faced a similar issue to that described at document-open time. When tags return to the document from the pcb after a paste operation, Lisp references to internal Lisp objects (e.g., to the hierarchy parent of a tag, to its corresponding end-tag if it is a start-tag, etc.) have been invalidated. (This would also happen with the native clipboard.) Thus, if the paste code determines that an element may be pasted, after doing so all its tags must have their references corrected in a manner similar to the corresponding problem at document-open time. However, in this case, searches can be restricted to the (usually small number of) tags in the text being pasted.

## 5 OVERLAPPING SPEECH AND TABLES

A characteristic of spontaneous dialogues is that speakers often overlap each other in time, and the precise nature of these overlaps is of interest to linguists for a number of reasons. For example, exactly how the floor is yielded may depend on what is being said and who is saying it.

Linguists have attempted many representation schemes to assist with this task. Elsewhere [6] we have characterized those schemes in two broad classes: those which are representationally complete in the sense that the overlap is algorithmically determinable from the representation and those which are visually faithful in the sense that it is easy to see what the overlap is. We wanted a scheme with both attributes, so we settled on a music-like notation, with vertical time axis.

Although some schemes support more precise timing indication, ICE markup demarks only the onset and end of overlap. Therefore, we do not need as much information as would be conveyed by music notation, and we found it adequate to use traditional tables, with some information conveyed by rule thickness. This allowed us to base our implementation on the Interleaf table package with only minimal modifications.

An example is in Appendix A. Each speaker’s turns are outlined with heavy rules. Each *overlapping string* (in ICE terminology) is outlined with a normal rule. Speech proceeds downward through the column, one column for each speaker. No reasonable table package will have consequential limits on the number of rows or columns, but of course if there are too many speakers reasonable display becomes impossible. In practice, there are seldom more speakers — say four or five — than fit on a turned page.

Several interesting problems arose for us in dealing with the tabular representation we chose. First, searching in Interleaf tables is row oriented, but in computing constraints we must search the current graph in column order, throughout the turn of a given speaker. Second, different things have to happen at the boundary of an overlapping string (i.e., a table cell), at a turn boundary (i.e., a cell whose top or bottom rule is thickened) and at the boundary of a table itself. To do this, we implemented a context sensitive search mechanism we describe in the next section.

The Interleaf table editor is quite easy to manipulate, but at this writing, we have not fully integrated the tabular representation shown in Appendix A with the tagging for speech described above. There are some interesting UI issues about whether tag insertion should be done on the fly as the table is interactively edited, on request of the user, upon exit from the table, closing the document, or perhaps upon some other event. Our user experience with the tabular representation is presently too limited to offer opinions on these questions.

## 6 CONTEXT SENSITIVE SEARCHING

Our solution to searching through a heterogeneous compound document consists of two constructs: a *context* for the search and a *continuation strategy*.

The context of a point in the document is a list of search context predicates which return true when applied to that point. These predicates come from a list, which includes things like *is-at-table-end, is-at-row-end, is at-column-end,* etc. Our present list of possible contexts is dedicated to searching in tables, but the mechanism can be used with any predicate expressible in Lisp. (Typically such predicates would answer questions about the given point in the document, but the mechanism is fully general.)

A continuation strategy is simply a Lisp function which accepts a point in the document and a direction and returns a *new* position in the document at which processing should continue given the original point and direction. For example, the strategy *table-continuation-strategy* returns a point in the first component of the next cell in the appropriate direction, whereas *table-exit-strategy* returns a point outside the table (if there is one). Because a context can have multiple predicates, arbitrarily complicated predicate logic can be performed to select a continuation strategy from the context. In our searches column-wise through tables this logic is very simple. However, one could imagine, say, selecting a particular strategy if a document point is in an equation and is in a graphics frame and is inside a circle containing the equation or is on an odd page . . . In general, the list of possible predicates and the logic for selecting a search continuation strategy from a context should be regarded as a property of a document or class of documents.

## 7 ICE, TEI, AND SGML

We started our project with the Text Encoding Initiative in mind, but as we focused on the ICE we spent more time on the needs defined by their markup scheme and less on the more daunting problem of general SGML solutions. ICE devised its own non-TEI (and therefore non-SGML) markup because (a) they wanted to proceed while TEI's work was still very much in progress, and (b) there is a perception among linguists that SGML is difficult and complex. However, some preliminary TEI-conformant DTDs for ICE markup have been written, and some such DTD is likely to become an official part of the ICE effort.

Accordingly, while our system is not restricted to ICE markup, it is not a general solution to the SGML tagging problem. In large part, this comes from our concept of category, which, although useful in this context and perhaps beyond, is a semantic notion that goes beyond what can be seen in most SGML applications.

In SGML terms a category can be thought of as an ‘or-group’ of elements. Any constraint on descendants (‘content model’) that allows one member of a category allows all of its members; and all members share a content model that allows 0 or more occurrences, in any order, of text or any member of the category’s child-categories.

However convenient we may have found categories, and however well they map to ICE’s ideas regarding the markup of speech transcriptions, most SGML DTDs are not ‘categorical’:

- aside from EMPTY or text-only elements, few element types share identical content models;
- sharing content models is rarely linked to interchangeability in the content models of other elements; and
- most content models are more prescriptive than the ones described above in terms of the order and number of elements needed for their satisfaction.

We expect that some part of our future work will be devoted to more robust SGML-compatibility.

On the other hand, some of what we’ve done is near the frontiers of SGML work to date. For instance, our notion of multiple graphs is basically the same as the SGML feature CONCUR. The feature itself is controversial within the SGML community, but the issues faced in grappling with it apply to any system that attempts to approach its functionality. In particular, although the TEI chose not to use CONCUR for its model of speech transcription markup, the scheme it chose presents the same problems to editing software as would a CONCUR-based scheme. Also, we’ve encountered constraints in ICE markup that are not easily expressible in SGML. For instance, overlapping strings are found only within speaker turns, but only when those speaker turns are themselves found inside a zone of overlap. There is no straightforward way to express this kind of constraint in SGML, depending as it does on both the parent and the grandparent of any given location in the text.

Meanwhile, the linguists’ perception of SGML as difficult and complex remains a stumbling block to its acceptance in this community. Part of the answer here lies in software like ours, which hides the syntactic complexity behind a task-oriented user interface. Another future endeavor for us will be an interactive constraint generator — a tool whereby linguists can develop tagging schemes to meet their needs without having to learn the complexities of SGML’s DTD syntax.

## 8 FUTURES

We plan to implement these features next:

- TEI DTDs
  In accordance with our initial goals, we plan to write software that accepts at least some TEI DTDs and generates our internal constraint representation, which is presently hand-generated. Although one of us (Blachman) has substantial SGML experience, we do not know how difficult this will be.
- Interactive constraint generator
  Although we believe TEI is likely to be widely adopted, many corpus linguists find SGML daunting, complex and obscure. The TEI documentation itself is tens of

times longer than that of the admittedly often naïve markup schemes invented by linguists with little experience of electronic document processing. This has led to TEI documentation which is precise and of high utility to computer scientists, but opaque and shunned by many linguists already using their own schemes. We hope to provide graphical tools by which a linguist can easily specify tag hierarchies which would automatically generate our constraint representation. Ideally, such software would in fact generate a TEI conformant DTD and, from that, our constraints.

- Automatic tagging, analysis tools
  Some tagging can be done semi-automatically with relative success, leaving the tagger to make only minor corrections. For example, sentence boundaries can conservatively be assumed to be any period-terminated text, leaving the user only to correct mistagged abbreviations and perhaps some numerical text. We have already written simple Lisp which traverses an Interleaf document and applies at every word boundary an arbitrary Lisp function given as argument, and we believe this may be sufficient for a lot of semi-automatic tagging.
  The same function provides a base for building interactive or batch analysis tools. Any linguistic question which can be answered by computations done at word boundaries will yield to this approach. As a trivial example one can implement a word counter in this way: the function is merely the incrementing at each word boundary of an integer variable. We intend to investigate whether more interesting tools can be built on this simple foundation.

## 9 SOME ABANDONED IDEAS

### 9.1 Incomplete elements

During most of our development, we were content to permit incomplete elements to exist in the document, that is, to allow a start-tag without its corresponding end. Here we are not referring to markup minimization in the cases where an end-tag is deducible from context. Rather, we implemented separate facilities for dropping start-tags and end-tags, the latter being offered only when there were start-tags with no corresponding end-tag. These are not fundamentally different constraint calculations, and the added code was only about a dozen lines. However, the complexities of computing reasonable cut and paste constraints in the face of unclosed tags led us to always drop a corresponding end-tag when we drop a start-tag, and to force each selection automatically to contain complete elements.

### 9.2 Milestones

Before we settled on the tabular representation of speech overlaps, we implemented a milestone approach. The user could drop numbered tags to indicate specific points of synchrony in each speaker's turn in a dialogue transcription. Although the software support was simple, it proved cumbersome for the tagger, who had no visual feedback to help find the appropriate milestone. Since the position of the milestone tags was arbitrary and determined by the amount of text between milestones, the tagger had to keep mental or paper track of the milestone id corresponding to a particular point in time.

## 10 SUMMARY

We have built a constraint-based interactive editor for linguistic markup which computes legal markup by analysing the current state of the document markup and comparing it to a constraint graph. To do this computation, we color the edges of multiple rooted constraint sub-graphs and take particular care at nodes lying in several colored subgraphs. To support searching through the document to verify constraints, we introduced a context-sensitive search mechanism based on predicates which are evaluated as the search comes to the boundaries of various pieces of the document structure.

## REFERENCES

1. H. Kucera and W.N. Francis, *Computational Analysis of Present-Day English*, Brown University Press, 1967.
2. C.M. Sperberg-McQueen and L. Burnard, ‘Guidelines for electronic text encoding’, Technical report, (1992). Available from the TEI Listserver (listserv@uicvm.bitnet).
3. G. Burnage and D. Dunlap, ‘Encoding the british national corpus’, 79–96, (1992).
4. S. Greenbaum, *A New Corpus of English: ICE*, 171–179, Lund University Press, 1980.
5. H. van Halteren and T. van den Heuvel, *Linguistic Exploitation of Syntactic Databases*, Rodopi, 1990.
6. C. Meyer, E.M. Blachman, and R.A. Morris, ‘Can you see whose speech is overlapping’, *Visible Language*. to appear.
7. D.D. Chamberlin, ‘Managing Properties in a System of Cooperating Editors’, in *Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography (EP90)*, ed., R. Furuta, volume 1, 31–46, Cambridge University Press, Cambridge, UK, (September 1990).
8. V. Quint and I. Vatton, ‘Grif: an interactive system for structured document manipulation’, in *Proceedings of the International Conference on Text Processing and Document Manipulation (EP86)*, ed., J.C. van Vliet, 200–213, Cambridge University Press, Cambridge, UK, (1986).
9. P.M. English, E. Jacobson, R.A. Morris, K.B. Mundy, S.D. Pelletier, T.A. Polucci, and H.D. Scarbro, ‘An Extensible, Object-Oriented System for Active Documents’, in *Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography (EP90)*, ed., R. Furuta, 263–276, Cambridge University Press, Cambridge, UK, (1990).
10. M. Hammer, R. Ilson, T. Anderson, E. Gilvert, M. Good, B. Naimir, L. Rosenstein, and S. Schoichet, *The implementation of Etude, an integrated and interactive document production system*, ACM Press, New York, 1981.

## APPENDIX A: TABLE REPRESENTATION OF OVERLAPPING SPEECH

| Bob | Ed | Chuck |
|---|---|---|
| Hello! I thought that I | | |
| might find | Hello. I've been | |
| | expecting you since this morning. | |
| Well, we had a power failure and I couldn't finish the manuscript. | | |
| I have a copy with me. | Well, that's not a prob-lem. I | |
| | have a copy here and we can work on that. Do you have it with | |
| But it is an old one, and | you? I didn't bring | |
| I think we had better start fresh. | | |
| | Good, let's start. | |
| Yes, let's | | |
| Hey, here's Chuck! Hi Chuck! | | |
| We want to meet | Hi, Chuck! | |
| the August 14th dead-line. Let's get to work | | |
| | | Hi guys, Sorry I'm late. We had a power failure. |
| You too? | | |
| | | Yes. |