Using SGML as a Basis for Data-Intensive NLP

David McKelvie, Chris Brew & Henry Thompson
Language Technology Group, Human Communication Research Centre,
University of Edinburgh, Edinburgh, Scotland
David.McKelvie@ed.ac.uk & Chris.Brew@ed.ac.uk & H.Thompson@ed.ac.uk

Abstract

This paper describes the LT NSL system (McKelvie et al, 1996), an architecture for writing corpus processing tools. This system is then compared with two other systems which address similar issues, the GATE system (Cunningham et al, 1995) and the IMS Corpus Workbench (Christ, 1994). In particular we address the advantages and disadvantages of an SGML approach compared with a non-SGML database approach.

1 Introduction

The theme of this paper is the design of software and data architectures for natural language processing using corpora. Two major issues in corpus-based NLP are: how best to deal with medium to large scale corpora often with complex linguistic annotations, and what system architecture best supports the reuse of software components in a modular and interchangeable fashion.

In this paper we describe the LT NSL system (McKelvie et al, 1996), an architecture for writing corpus processing tools, which we have developed in an attempt to address these issues. This system is then compared with two other systems which address some of the same issues, the GATE system (Cunningham et al, 1995) and the IMS Corpus Workbench (Christ, 1994). In particular we address the advantages and disadvantages of an SGML approach compared with a non-SGML database approach. Finally, in order to back up our claims about the merits of SGML-based corpus processing, we present a number of case studies of the use of the LT NSL system for corpus preparation and linguistic analysis.

2 The LT NSL system

LT NSL is a tool architecture for SGML-based processing of (primarily) text corpora. It generalises the UNIX pipe architecture, making it possible to use pipelines of general-purpose tools to process annotated corpora. The original UNIX architecture allows the rapid construction of efficient pipelines of conceptually simple processes to carry out relatively complex tasks, but is restricted to a simple model of streams as sequences of bytes, lines or fields. LT NSL lifts this restriction, allowing tools access to streams which are sequences of tree-structured text (a representation of SGML marked-up text).

The use of SGML as an I/O stream format between programs has the advantage that SGML is a well defined standard for representing structured text. Its value is precisely that it closes off the option of a proliferation of ad-hoc notations and the associated software needed to read and write them. The most important reason why we use SGML for all corpus linguistic annotation is that it forces us to formally describe the markup we will be using and provides software for checking that these markup invariants hold in an annotated corpus. In practise this is extremely useful. SGML is human readable, so that intermediate results can be inspected and understood. It also means that it is easy for programs to access the information which is relevant to them, while ignoring additional markup. A further advantage is that many text corpora are available in SGML, for example, the British National Corpus (Burnage&Dunlop, 1992).

The LT NSL system is released as C source code. The software consists of a C-language Application Program Interface (API) of function calls, and a number of stand-alone programs which use this API. The current release is known to work on UNIX (SunOS 4.1.3, Solaris 2.4 and Linux), and a Windows-NT version will be released during 1997. There is also an API for the Python programming language.

One question which arises in respect to using SGML as an I/O format is: what about the cost of

parsing SGML? Surely that makes pipelines too inefficient? Parsing SGML in its full generality, and providing validation and adequate error detection is indeed rather hard. For efficiency reasons, you wouldn't want to use long pipelines of tools, if each tool had to reparse the SGML and deal with the full language. Fortunately, LT NSL doesn't require this. The first stage of processing *normalises* the input, producing a simplified, but informationally equivalent form of the document. Subsequent tools can and often will use the LT NSL API which parses normalised SGML (henceforth NSGML) approximately ten times more efficiently than the best parsers for full SGML. The API then returns this parsed SGML to the calling program as data-structures.

NSGML is a fully expanded text form of SGML informationally equivalent to the ESIS output of SGML parsers. This means that all markup minimisation is expanded to its full form, SGML entities are expanded into their value (except for SDATA entities), and all SGML names (of elements, attributes, etc) are normalised. The result is a format easily readable by humans and programs.

The LT NSL programs consist of `mknsg`, a program for converting arbitrary valid SGML into normalised SGML[1], the first stage in a pipeline of LT NSL tools; and a number of programs for manipulating normalised SGML files, such as `sggrep` which finds SGML elements which match some query. Other of our software packages such as LT POS (a part of speech tagger) and LT WB (Mikheev&Finch, 1997) also use the LT NSL library.

In addition to the normalised SGML, the `mknsg` program writes a file containing a compiled form of the Document Type Definition (DTD)[2], which LT NSL programs read in order to know what the structure of their NSGML input or output is.

How fast is it? Processes requiring sequential access to large text corpora are well supported. It is unlikely that LT NSL will prove the rate limiting step in sequential corpus processing. The kinds of repeated search required by lexicographers are more of a problem, since the system was not designed for that purpose. The standard distribution is fast enough for use as a search engine with files of up to several million words. Searching 1% of the British National Corpus (a total of 700,000 words (18 Mb)) is currently only 6 times slower using LT NSL `sggrep` than using `fgrep`, and `sggrep` allows more complex structure-sensitive queries. A prototype indexing mechanism (Mikheev&McKelvie, 1997), not yet in the distribution, improves the performance of LT NSL to acceptable levels for much larger datasets.

Why did we say "primarily for text corpora"? Because much of the technology is directly applicable to multimedia corpora such as the Edinburgh Map Task corpus (Anderson et al, 1991). There are tools which interpret SGML elements in the corpus text as offsets into files of audio-data, allowing very flexible retrieval and output of audio information using queries defined over the corpus text and its annotations. The same could be done for video clips, etc.

2.1 Hyperlinking

We are inclined to steer a middle course between a monolithic comprehensive view of corpus data, in which all possible views, annotations, structurings etc. of a corpus component are combined in a single heavily structured document, and a massively decentralised view in which a corpus component is organised as a hyper-document, with all its information stored in separate documents, utilising inter-document pointers. Aspects of the LT NSL library are aimed at supporting this approach. It is necessary to distinguish between *files*, which are storage units, *(SGML) documents*, which may be composed of a number of files by means of external entity references, and *hyper-documents*, which are linked ensembles of documents, using e.g. HyTime or TEI (Sperberg-McQueen&Burnard, 1994) link notation.

The implication of this is that corpus components can be hyper-documents, with low-density (i.e. above the token level) annotation being expressed indirectly in terms of links. In the first instance, this is constrained to situations where element content at one level of one document is entirely composed of elements from another document. Suppose, for example, we already had segmented a file resulting in a single document marked up with SGML headers and paragraphs, and with the word segmentation marked with <w> tags:

```
. . .
<p id=p4>
<w id=p4.w1>Time</w>
<w id=p4.w2>flies</w>
<w id=p4.w3>.</w>
</p>
. . .
```

The output of a phrase-level segmentation might then be stored as follows:

```
. . .
<p id=p4>
<phr id=p4.ph1 type=n doc=file1 from='id p4.w1'>
<phr id=p4.ph2 type=v from='id p4.w2'>
</p>
. . .
```

[1] Based on James Clark's SP parser (Clark, 1996).

[2] SGML's way of describing the structure (or grammar) of the allowed markup in a document

Linking is specified using one of the available TEI mechanisms. Details are not relevant here, suffice it to say that `doc=file1` resolves to the word level file and establishes a default for subsequent links. At a minimum, links are able to target single elements or sequences of contiguous elements. LT NSL implements a *textual inclusion* semantics for such links, inserting the referenced material as the content of the element bearing the linking attributes. Although the example above shows links to only one document, it is possible to link to several documents, e.g. to a word document and a lexicon document:

```
<word>
<source doc=file1 from='id p4.w1'>
<lex doc=lex1 from='id lex.40332'>
</word>
```

Note that the architecture is recursive, in that e.g. sentence-level segmentation could be expressed in terms of links into the phrase-level segmentation as presented above.

The data architecture needs to address not only multiple levels of annotation but also alternative versions at a given level. Since our linking mechanism uses the SGML entity mechanism to implement the identification of target documents, we can use the entity manager's catalogue as a means of managing versions. For our example above, this means that the connection between the phrase encoding document and the segmented document would be in two steps: the phrase document would use a PUBLIC identifier, which the catalogue would map to the particular file. Since catalogue entries are interpreted by tools as local to the directory where the catalogue itself is found, this means that binding together groups of alternative versions can be easily achieved by storing them under the same directory.

Subdirectories with catalogue fragments can thus be used to represent both increasing detail of annotation and alternatives at a given level of annotation.

Note also that with a modest extension of functionality, it is possible to use the data architecture described here to implement patches, e.g. to the tokenisation process. If alongside an inclusion semantics, we have a special empty element <repl> which is *replaced* by the range it points to, we can produce a patch file, e.g. for a misspelled word, as follows (irrelevant details omitted):

```
<nsl>
<!-- to get the original header-->
<repl doc=original from='id hdr1'>
<text>
<!-- the first swatch of unchanged text -->
<repl from='id p1' to='id p324'>
<!-- more unchanged text -->
<p id=p325>
<repl from='id p325.t1' to='id p325.t15'>
<!-- the correction itself -->
<corr sic='procede' resp='ispell'>
<token id=p325.t16>proceed</token>
</corr>
<!-- more unchanged text-->
<repl from='id p325.t17' to='id p325.t96'>
</p>
<!-- the rest of the unchanged text-->
<repl from='id p326' to='id p402'>
</text>
</nsl>
```

Whether such a patch would have knock-on effects on higher levels of annotation would depend, inter alia, on whether a change in tokenisation crossed any higher-level boundaries.

2.2 sggrep and the LT NSL query language

The API provides the program(mer) with two alternative views of the NSGML stream: an object stream view and a tree fragment view. The first, lower level but more efficient, provides data structures and access functions such as `GetNextBit` and `PrintBit`, where there are different types of `Bits` for start (or empty) tags with their attributes, text content, end tags, and a few other bits and pieces.

The alternative, higher level, view, lets one treat the NSGML input as a sequence of tree-fragments. The API provides functions `GetNextItem` and `PrintItem` to read and write the next complete SGML element. It also provides functionality `GetNextQueryElement(infile,query,subquery, regexp,outfile)` where `query` is an LT NSL query which allows one to specify particular elements on the basis of their position in the document structure and their attribute values. The `subquery` and `regexp` allow one to specify that the matching element has a subelement matching the subquery with text content matching the regular expression. Elements which do not match the query are passed through unchanged to `outfile`. Under both models, processing is essentially a loop over calls to the API, in each case choosing to discard, modify or output unchanged each `Bit` or `Element`.

Rather than define the query language here (details can be found in (McKelvie et al, 1996)), we will just provide an example. The call

```
GetNextQueryElement(inf,".*/TEXT/.*/P",
                    "P/.*/S","th(ei|ie)r", outf)
```

would return the next <P> element dominated anywhere by <TEXT> at any depth, with the <P> element satisfying the additional requirement that it contain at least one <S> element at any depth with text containing at least one instance of 'their' (possibly misspelt).

3 Comparisons with other systems

The major alternative corpus architecture which has been advocated is a database approach, where annotations are kept separately from the base texts. The annotations are linked to the base texts either by means of character offsets or by a more sophisticated indexing scheme. We will discuss two such systems and compare them with the LT NSL approach.

3.1 GATE

The GATE system (Cunningham et al, 1995), currently under development at the University of Sheffield, is a system to support modular language engineering.

3.1.1 System components

It consists of three main components:

- GDM - an object oriented database for storing information about the corpus texts. This database is based on the TIPSTER document architecture (Grishman, 1995), and stores text annotations separate from the texts. Annotations are linked to texts by means of character offsets[3].
- Creole - A library of program and data resource wrappers, that allow one to interface externally developed programs/resources into the GATE architecture.
- GGI - a graphical tool shell for describing processing algorithms and viewing and evaluating the results.

A MUC-6 compatible information extraction system, VIE, has been built using the GATE architecture.

3.1.2 Evaluation

Separating corpus text from annotations is a general and flexible method of describing arbitrary structure on a text. It may be less useful as a means of publishing corpora and may prove inefficient if the underlying corpus is liable to change.

Although TIPSTER lets one define annotations and their associated attributes, in the present version (and presumably also in GATE) these definitions are treated only as documentation and are not validated by the system. In contrast, the SGML parser validates its DTD, and hence provides some check that annotations are being used in their intended way. SGML has the concept of *content models* which restrict the allowed positions and nesting of annotations. GATE allows any annotation anywhere. Although this is more powerful, i.e. one is not restricted to tree structures, it does make validation of annotations more difficult.

The idea of having formalised interfaces for external programs and data is a good one.

The GGI graphical tool shell lets one build, store, and recover complex processing specifications. There is merit in having a high level language to specify tasks which can be translated automatically into executable programs (e.g. shell scripts). This is an area that LT NSL does not address.

3.1.3 Comparison with LT NSL

In (Cunningham et al, 1996), the GATE architecture is compared with the earlier version of the LT NSL architecture which was developed in the MULTEXT project. We would like to answer these points with reference to the latest version of our software.

It is claimed that using normalised SGML implies a large storage overhead. Normally however, normalised SGML will be created on the fly and passed through pipes and only the final results will need to be stored. This may however be a problem for very large corpora such as the BNC.

It is stated that representing ambiguous or overlapping markup is complex in SGML. We do not agree. One can represent overlapping markup in SGML in a number of ways. As described above, it is quite possible for SGML to represent 'stand-off' annotation in a similar way to TIPSTER. LT NSL provides the hyperlinking semantics to interpret this SGML.

The use of normalised SGML and a compiled DTD file means that the overheads of parsing SGML in each program are small, even for large DTDs, such as the TEI.

LT NSL is not specific to particular applications or DTDs. The MULTEXT architecture was tool-specific, in that its API defined a predefined set of abstract units of linguistic interest, words, sentences, etc. and defined functions such as `ReadSentence`. That was because MULTEXT was undecided about the format of its I/O. LT NSL in contrast, since we have decided on SGML as a common format, provides functions such as `GetNextItem` which read the next SGML element. Does this mean the LT NSL architecture is application neutral? Yes and no.

Yes, because there is in principle no limit on what can be encoded in an SGML document. In the TIPSTER architecture there is an architectural requirement that all annotations be ultimately associated with spans of a single base text, but LT NSL imposes

[3] More precisely, by inter byte locations.

no such requirement. This makes it easier to be clear about what happens when a different view is needed on fixed-format read-only information, or when it turns out that the read-only information should be systematically corrected. The details of this are a matter of ongoing research, but an important motivation for the architecture of LT NSL is to allow such edits without requiring that the read-only information be copied.

No, because in practice any corpus is encoded in a way which reflects the assumptions of the corpus developers. Most corpora include a level of representation for words, and many include higher level groupings such as breath groups, sentences, paragraphs and/or documents. The sample back-end tools distributed with LT NSL reflect this fact.

It is claimed that there is no easy way in SGML to differentiate sets of results by who or what produced them. But, to do this requires only a convention for the encoding of *meta-information* about text corpora. For example, SGML DTDs such as the TEI include a 'resp' attribute which identifies who was responsible for changes. LT NSL does not require tools to obey any particular conventions for meta-information, but once a convention is fixed upon it is straightforward to encode the necessary information as SGML attributes.

Unlike TIPSTER, LT NSL is not built around a database, so we cannot take advantage of built-in mechanisms for version control. As far as corpus annotation goes, UNIX rcs, has proved an adequate solution to our version control needs. Alternatively, version control can be provided by means of hyperlinking.

The GATE idea of providing formal wrappers for interfacing programs is a good one. In LT NSL the corresponding interfaces are less formalised, but can be defined by specifying the DTDs of a program's input and output files. For example a part-of-speech tagger would expect <W> elements inside <S> elements, and a 'TAG' attribute on the output <W> elements. Any input file whose DTD satisfied this constraint could be tagged. SGML architectural forms (a method for DTD subsetting) could provide a method of formalising these program interfaces.

As Cunningham et. al. say, there is no reason why there could not be an implementation of LT NSL which read SGML elements from a database rather than from files. Similarly, a TIPSTER architecture like GATE could read SGML and convert it into its internal database. In that case, our point would be that SGML is a suitable abstraction for programs rather than a more abstract (and perhaps more limited) level of interface. We are currently in discussion with the GATE team about how best to allow the interoperability of the two systems.

3.2 The IMS Corpus Workbench

The IMS Corpus Workbench (Christ, 1994) includes both a query engine (CQP) and a Motif-based user visualisation tool (*xkwic*). CQP provides a query language which is a conservative extension of familiar UNIX regular expression facilities[4]. XKWIC is a user interface tuned for corpus search. As well as providing the standard keyword-in-context facilities and giving access to the query language it gives the user sophisticated tools for managing the query history, manipulating the display, and storing search results. The most interesting points of comparison with LT NSL are in the areas of query language and underlying corpus representation.

3.2.1 The CQP model

CQP treats corpora as sequences of attribute-value bundles. Each attribute[5] can be thought of as a total function from corpus positions to attribute values. Syntactic sugar apart, no special status is given to the attribute `word`.

3.2.2 The query language

The query language of IMS-CWB, which has the usual regular expression operators, works uniformly over both attribute values and corpus positions. This regularity is a clear benefit to users, since only one syntax must be learnt.

Expressions of considerable sophistication can be generated and used successfully by beginners. Consider:

```
[pos="DT" & word !="the"] [pos="JJ.*"]?
[pos="N.+"]
```

This means, in the context of the Penn treebank tagset, "Find me sequences beginning with determiners other than *the*, followed by optional adjectives, then things with nominal qualities". The intention is presumably to find a particular sub-class of noun-phrases.

The workbench has plainly achieved an extremely successful generalisation of regular expressions, and one which has been validated by extensive use in lexicography and corpus-building.

There is only limited access to structural information. While it is possible, if sentence boundaries are marked in the corpus, to restrict the search to

[4]Like LT NSL IMS-CWB is built on top of Henry Spencer's public domain regular expression package

[5]In CQP terminology these are the "positional attributes".

within-sentence matches, there are few facilities for making more refined use of hierarchical structure. The typical working style, if you are concerned with syntax, is to search for sequences of attributes which you believe to be highly correlated with particular syntactic structures.

3.2.3 Data representation

CQP requires users to transform the corpora which will be searched into a fast internal format. This format has the following properties:

- Because of the central role of corpus position it is necessary to tokenise the input corpus, mapping each word in the raw input to a set of attribute value pairs and a corpus position.
- There is a logically separate index for each attribute name in the corpus.
- CQP uses an integerised representation, in which corpus items having the same value for an attribute are mapped into the same integer descriptor in the index which represents that attribute. This means that the character data corresponding to each distinct corpus token need only be stored once.
- For each attribute there is an item list containing the sequence of integer descriptors corresponding to the sequence of words in the corpus. Because of the presence of this list the storage cost of adding a new attribute is linear in the size of the corpus. If the new attribute were sparse, it would be possible to reduce the space cost by switching (for that attribute) to a more space efficient encoding[6]

3.2.4 Evaluation

The IMS-CWB is a design dominated by the need for frequent fast searches of a corpus with a fixed annotation scheme. Although disk space is now cheap, the cost of preparing and storing the indices for IMS-CWB is such that the architecture is mainly appropriate for linguistic and lexicographic exploration, but less immediately useful in situations, such as obtain in corpus development, where there is a recurring need to experiment with different or evolving attributes and representational possibilities.

Some support is provided for user-written tools, but as yet there is no published API to the potentially very useful query language facilities. The indexing tools which come with IMS-CWB are less flexible than those of LT NSL since the former must index on words, while the latter can index on any level of the corpus annotation.

The query language of IMS-CWB is an elegant and orthogonal design, which we believe it would be appropriate to adopt or adapt as a standard for corpus search. It stands in need of extension to provide more flexible access to hierarchical structure[7]. The query language of LT NSL is one possible template for such extensions, as is the opaque but powerful `tgrep` program (Pito, 1994) which is provided with the Penn Treebank.

4 Case studies

4.1 Creation of marked-up corpora

One application area where the paradigm of sequential adding of markup to an SGML stream fits very closely, is that of the production of annotated corpora. Marking of major sections, paragraphs and headings, word tokenising, sentence boundary marking, part of speech tagging and parsing are all tasks which can be performed sequentially using only a small moving window of the texts. In addition, all of them make use of the markup created by earlier steps. If one is creating an annotated corpus for public distribution, then SGML is (probably) the format of choice and thus an SGML based NLP system such as LT NSL will be appropriate.

Precursors to the LT NSL software were used to annotate the MLCC corpora used by the MULTEXT project. Similarly LT NSL has been used to recode the Edinburgh MapTask corpus into SGML markup, a process which showed up a number of inconsistencies in the original (non-SGML) markup. Because LT NSL allows the use of multiple I/O files (with different DTDs), in (Brew&McKelvie, 1996) it was possible to apply these tools to the task of finding translation equivalencies between English and French. Using part of the MLCC corpus, part-of-speech tagged and sentence aligned using LT NSL tools, they explored various techniques for finding word alignments. The LT NSL programs were useful in evaluating these techniques. See also (Mikheev&Finch, 1995), (Mikheev&Finch, 1997) for other uses of the LT NSL tools in annotating linguistic structures of interest and extracting statistics from that markup.

4.2 Transformation of corpus markup

Although SGML is human readable, in practice once the amount of markup is of the same order of magni-

[6] IMS-CWB already supports compressed index files, and special purpose encoding formats would presumably save even more space.

[7] This may be a specialised need of academic linguists, and for many applications it is undoubtedly more important to provide clean facilities for non-hierarchical queries but it seems premature to close off the option of such access.

tude as the textual content, reading SGML becomes difficult. Similarly, editing such texts using a normal text editor becomes tedious and error prone. Thus if one is committed to the use of SGML for corpus-based NLP, then one needs to have specialised software to facilitate the viewing and editing of SGML. A similar problem appears in the database approach to corpora, where the difficulty is not in seeing the original text, but in seeing the markup in relationship to the text.

4.2.1 Batch transformations

To address this issue LT NSL includes a number of text based tools for the conversion of SGML: `textonly`, `sgmltrans` and `sgrpg`. With these tools it is easy to select portions of text which are of interest (using the query language) and to convert them into either plain text or another text format, such as LaTeX or HTML. In addition, there are a large number of commercial and public domain software packages for transforming SGML. In the future, however, the advent of the DSSSL transformation language will undoubtably revolutionise this area.

4.2.2 Hand correction

Specialised editors for SGML are available, but they are not always exactly what one wants, because they are *too* powerful, in that they let all markup and text be edited. What is required for markup correction are specialised editors which only allow a specific subset of the markup to be edited, and which provide an optimised user interface for this limited set of edit operations.

In order to support the writing of specialised editors, we have developed a Python (vanRossum, 1995) API for LT NSL, (Tobin&McKelvie, 1996). This allows us to rapidly prototype editors using the Python/Tk graphics package. These editors can fit into a pipeline of LT NSL tools allowing hand correction or disambiguation of markup automatically added by previous tools. Using this API we are developing a generic SGML editor. It is an object-oriented system where one can flexibly associate display and interaction classes to particular SGML elements. Already, this generic editor has been used for a number of tasks; the hand correction of part-of-speech tags in the MapTask, the correction of turn boundaries in the Innovation corpus (Carletta et al, 1996), and the evaluation of translation equivalences between aligned multilingual corpora.

We found that using this generic editor framework made it possible to quickly write new editors for new tasks on new corpora.

5 Conclusions

SGML is a good markup language for base level annotations of published corpora. Our experience with LT NSL has shown that:

- It is a good system for sequential corpus processing where there is locality of reference.
- It provides a modular architecture which does not require a central database, thus allowing distributed software development and reuse of components.
- It works with existing corpora without extensive pre-processing.
- It does support the Tipster approach of separating base texts from additional markup by means of hyperlinks. In fact SGML (HyTime) allows much more flexible addressing, not just character offsets. This is of benefit when working with corpora which may change.

LT NSL is not so good for:

- Applications which require a database approach, i.e. those which need to access markup at random from a text, for example lexicographic browsing or the creation of book indexes.
- Processing very large plain text or unnormalised SGML corpora, where indexing is required, and generation of normalised files is a large overhead. We are working on extending LT NSL in this direction, e.g. to allow processing of the BNC corpus in its entirety.

In conclusion, the SGML and database approaches are optimised for different NLP applications and should be seen as complimentary rather than as conflicting. There is no reason why one should not attempt to use the strengths of both the database and the SGML stream approaches. It is recommended that future work should include attention to allowing interfacing between both approaches.

6 Acknowledgements

This work was carried out at the Human Communication Research Centre, whose baseline funding comes from the UK Economic and Social Research Council. The LT NSL work began in the context of the LRE project MULTEXT with support from the European Union. It has benefited from discussions with other MULTEXT partners, particularly ISSCO Geneva, and drew on work at our own institution by

Steve Finch and Andrei Mikheev. We also wish to thank Hamish Cunningham and Oliver Christ for useful discussions.

References

A. H. Anderson, M. Bader, E. G. Bard, E. H. Boyle, G. M. Doherty, S. C. Garrod, S. D. Isard, J. C. Kowtko, J. M. McAllister, J. Miller, C. F. Sotillo, H. S. Thompson, and R. Weinert. The HCRC Map Task Corpus. *Language and Speech*, 34(4):351–366, 1991.

C. Brew and D. McKelvie. 1996. "Word-pair extraction for lexicography". In *Proceedings of NeMLaP'96*, pp 45-55, Ankara, Turkey.

G. Burnage and D. Dunlop. 1992. "Encoding the British National Corpus". In *13th International Conference on English Language research on computerised corpora*, Nijmegen. Available at `http://www.sil.org/sgml/bnc-encoding2.html` See also `http://info.ox.ac.uk/bnc/`

J. Carletta, H. Fraser-Krauss and S. Garrod. 1996. "An Empirical Study of Innovation in Manufacturing Teams: a preliminary report". In *Proceedings of the International Workshop on Communication Modelling (LAP-96)*, ed. J. L. G. Dietz, Springer-Verlag, Electronic Workshops in Computing Series.

O. Christ. 1994. "A modular and flexible architecture for an integrated corpus query system". In *Proceedings of COMPLEX '94: 3rd Conference on Computational Lexicography and Text Research* (Budapest, July 7-10, 1994), Budapest, Hungary. CMP-LG archive id 9408005

J. Clark. 1996 "SP: An SGML System Conforming to International Standard ISO 8879 – Standard Generalized Markup Language". Available from `http://www.jclark.com/sp/index.htm`.

H. Cunningham, Y. Wilks and R. J. Gaizauskas. 1996. "New Methods, Current Trends and Software Infrastructure for NLP". In *Proceedings of the Second Conference on New Methods in Language Processing*, pages 283–298, Ankara, Turkey, March.

H. Cunningham, R. Gaizauskas and Y. Wilks. 1995. "A General Architecture for Text Engineering (GATE) - a new approach to Language Engineering R&D". *Technical Report, Dept of Computer Science, University of Sheffield.* Available from `http://www.dcs.shef.ac.uk/research/groups/nlp/gate/`

R. Grishman. 1995. "TIPSTER Phase II Architecture Design Document Version 1.52". *Technical Report, Dept. of Computer Science, New York University.* Available at `http://www.cs.nyu.edu/tipster`

D. McKelvie, H. Thompson and S. Finch. 1996. "The Normalised SGML Library LT NSL version 1.4.6". *Technical Report, Language Technology Group, University of Edinburgh.* Available at `http://www.ltg.ed.ac.uk/software/nsl`

A. Mikheev and S. Finch. 1995. "Towards a Workbench for Acquisition of Domain Knowledge from Natural Language". In *Proceedings of the Seventh Conference of the European Chapter of the Association for Computational Linguistics (EACL'95).* Dublin, Ireland.

A. Mikheev and S. Finch. 1997. "A Workbench for Finding Structure in Texts". in these proceedings.

A. Mikheev and D. McKelvie. 1997. "Indexing SGML files using LT NSL". *Technical Report, Language Technology Group, University of Edinburgh.*

R. Pito. 1994. "Tgrep Manual Page". Available from `http://www.ldc.upenn.edu/ldc/online/treebank/man/`

G. van Rossum. 1995. "Python Tutorial". Available from `http://www.python.org/`

C. M. Sperberg-McQueen & L. Burnard, eds. 1994. "Guidelines for Electronic Text Encoding and Interchange". Text Encoding Initiative, Oxford.

R. Tobin and D. McKelvie. 1996. "The Python Interface to the Normalised SGML Library (PythonNSL)". *Technical Report, Language Technology Group, University of Edinburgh.*