Proceedings of the Eighth European Conference on Artificial Intelligence (ECAI-98), 1998, Brighton, UK pp 185-189.

Sense Tagging and Language Engineering

Mark Stevenson, Hamish Cunningham and Yorick Wilks[1]

Abstract. Language engineering is a new, practical, branch of NLP and Computational Linguistics. We describe two language engineering systems: GATE (General Architecture for Text Engineering), an architecture to aid in the production and delivery of language engineering systems which significantly reduces development time and ease of reuse. We also describe a sense tagger which we implemented within the GATE architecture, and which achieves high accuracy (93% of all words in text to a broad semantic level). The implementation of the sense tagger was used as a practical task on which to evaluate the usefulness of the GATE architecture and identify its strengths and weaknesses.

1 Introduction

Within the last few years a new area has emerged from the fields of Computational Linguistics and Natural Language Processing: Language Engineering. This has been defined in various ways by different authors [2, 11], however the basic principal is that Language Engineering is the discipline or act of engineering software systems that perform tasks involving processing human language. The literature of the field relates to both application of relevant scientific results and to a body of practice. Language Engineering deals with practical approaches to real problems it does not, and indeed cannot, restrict itself to the sort of toy domains early NLP systems were restricted to (see [23]).

In this paper we explore two aspects of this new area: a system designed to aid the Language Engineer, GATE (General Architecture for Text Engineering) [8], and a novel hybrid sense tagger for unrestricted text [24], a language engineering application developed within this architecture. Both topics are relevant to current research: the first because work on NLP components shares a heap of problems with other artificial intelligence to do with reusability, data visualisation, and software-level robustness and efficiency that, we feel, are best solved by the provision of a general architectures and development environments; the second because both the problem of arriving at methods for sense tagging is accepted as important if language processing applications are ever to be useful in general domains [22].

We begin with a review of the GATE document processing architecture (section 2). The sense tagging problem is explained (section 3) and our system described (section 4). We conclude with some observations drawn from this work (section 5).

[1] Department of Computer Science, University of Sheffield, Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK {marks, hamish, yorick}@dcs.shef.ac.uk

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

2 GATE

GATE is an architecture and development environment for research and development workers in NLP and Language Engineering. It is an architecture in the sense that it specifies a macro-level organisational pattern for the various components and data resources that make up a language processing (actually at present only *text* processing) system [19]. It is also a development environment that adds a rich set of graphical tools to the architecture enabling the developer to easily integrate new processing components, to manage flow of control between components, to visualise the data produced by and passed between components, and evaluate the contribution of components to some externally defined and measured language processing task.

The motivating factors behind development of the architecture included the facilitation of reuse of components (which has previously been successful in NLP only for data resources [6, 7]), comparative and task-based evaluation, collaborative research, and software-level robustness, efficiency and portability. The design we arrived at in support of these aims is sketched in the rest of this section.

NLP systems produce information about texts (which may sometimes be the results of automatic speech recognition) and existing systems that aim to provide software infrastructure for NLP can be classified as belonging to one of three types according to the way in which they treat this information:

additive, or markup-based: information produced is added to the text in the form of markup, e.g. in SGML [21];

referential, or annotation-based: information is stored separately with references back to the original text, e.g. in the TIPSTER architecture [12];

abstraction-based: the original text is preserved in processing only as parts of an integrated data structure that represents information about the text in a uniform theoretically-motivated model, e.g. attribute-value structures in the ALEP system [20].

A fourth category might be added to cater for those systems that provide communication and control infrastructure without addressing the text-specific needs of NLP (e.g. Verbmobil's ICE architecture [1]).

We believe that performance and other considerations favour the referential approach, but also that SGML is a key part of any general text processing strategy. The first design decision we made, then, was to base GATE on a referential core using the TIPSTER architecture, and to cater for SGML via I/O format conversion filters. This led to the development of one of three key pillars of the system: GDM, the GATE Document Manager. GDM and the TIPSTER API that it implements forms a buffer between processing modules in a GATE-based NLP system. Modules no longer talk to each other, with the coherence and coupling implications that direct unrestricted communication can imply, but to GDM via the TIPSTER API.

One of the key benefits of adopting an explicit architecture for data management is that it becomes possible to easily add a of layer graphical interface access to architectural services and data visualisation tools, and such a layer is our second pillar: GGI, the GATE Graphical Interface. GGI has functions for creating, viewing and editing the collections of documents which are managed by the GDM and that form the corpora which LE modules and systems in GATE use as input data. It also has facilities to display the results of module or system execution – new or changed annotations associated with the document. These annotations can be viewed either in raw form, using a generic annotation viewer, or in an annotation-specific way, if special annotation viewers are available. For example, named entity annotations which identify and classify proper names (e.g. organization names, person names, location names) are shown by colour-coded highlighting of relevant words; phrase structure annotations are shown by graphical presentation of parse trees. Note that the viewers are general for particular types of annotation, so, for example, the same procedure is used for any POS tag set, Named-Entity markup etc. Thus developers reuse GATE data visualisation code with negligible overhead.

Lastly, the third pillar of the system is the one that does all the real work of processing texts and discovering information about their content: CREOLE, a Collection of REusable Objects for Language Engineering. In a sense CREOLE isn't part of GATE at all, but is the set of resources currently integrated with the system, but we also use the term to refer to the mechanisms available for integrating modules into GATE. This process has been automated to a large degree and can be driven from the interface. The developer is required to produce some C++ or Tcl code that uses the GDM TIPSTER API to get information from the database and write back results. When the module pre-dates integration, this is called a *wrapper* as it encapsulates the module in a standard form that GATE expects. When modules are developed specifically for GATE they can embed TIPSTER calls throughout their code and dispense with the wrapper intermediary. The underlying module can be an external executable written in any language (the current CREOLE set includes Prolog, Lisp and Perl programs, for example).

CREOLE wrappers encapsulate information about the preconditions for a module to run (data that must be present in the GDM database) and post-conditions (data that will result). This information is needed by GGI, and is provided by the developer in a configuration file, which also details what sort of viewer to use for the module's results and any parameters that need passing to the module. These parameters can be changed from the interface at run-time, e.g. to tell a parser to use a different lexicon. Aside from the information needed for GGI to provide access to a module, GATE compatibility equals TIPSTER compatibility – i.e. there will be very little overhead in making any TIPSTER module run in GATE. Given an integrated module, all other interface functions happen automatically. For example, the module will appear in a graph of all modules available, with permissible links to other modules automatically displayed, having been derived from the module pre- and post-conditions. At any point the developer can create a new graph from a subset of available CREOLE modules to perform a task of specific interest.

The integration mechanisms also reduce the documentation load: users can reference the TIPSTER API to describe the interchange format of the data they produce and the GATE documentation for integration details. Of course GATE doesn't solve all the problems involved in plugging diverse LE modules together. There are three barriers to such integration:

- managing *storage and exchange* of information about texts;
- incompatibility of *representation* of information about texts;
- incompatibility of *type* of information used and produced by different modules.

GATE provides a solution to the first two of these, allowing the integrator to concentrate on the core issue of the meaningful content of the information exchanged.

3 Word sense tagging

Sense tagging is the process of assigning the appropriate sense from some semantic lexicon to all ambiguous words in natural text.[2] This is similar to the more widely known technology of part-of-speech tagging, but the tags which are assigned in sense tagging are semantic tags from a dictionary rather than the grammatical tags assigned by a part-of-speech tagger.

Our sense tagger uses the machine readable version of *Longman Dictionary of Contemporary English* (LDOCE) [17] to provide the semantic tag set. LDOCE is a learners' dictionary – one designed not for native speakers of English but for those learning English as a second language.

Sense tagging should not be confused with the more common language processing task of *word sense disambiguation* (see [16], [18] and [27]). There are two main differences between sense tagging and word sense disambiguation: word sense disambiguation algorithms tend to limit themselves to a small number of word types, for example Yarowskys unsupervised word sense disambiguation system disambiguated only 12 different words. Another difference is that the tags assigned in semantic tagging are taken from a standard lexicon in which words may be many-way polysemous (for example "book" has 14 senses in LDOCE) while most word sense disambiguation algorithms assume words to have only two senses. Sense tagging can be considered as a true language engineering task since it is concerned with the robust processing of large amounts of text and aims to provide useful information for further language engineering processing while word sense disambiguation has more limited aims and is mainly experimental in nature.

4 The Sense Tagger

An interesting fact about recent semantic disambiguation algorithms is that they have made use of different, orthogonal, sources of information: the information provided by each source seems independent of and has no bearing on any of the others. For example, Yarowsky used syntactic collocates [26], Mahesh et. al. [14] used selectional restrictions and Bruce and Guthrie used pragmatic codes from LDOCE [4], in these different

[2] This is often loosened to each *content* word.

methods we can see that syntactic, semantic and pragmatic information has been used in word sense disambiguation. Each of these knowledge sources has enjoyed some degree of success which would suggest that they all have something to add to the problem of resolving senses in texts. A natural extension to this observation is to create a disambiguation system which makes use of several of these independent knowledge sources and combines their results in an intelligent way.

Our system is based on a set of *partial taggers*, each of which uses a different knowledge source, with their results being combined. Our system is in the tradition of McRoy [15], who also made use of several knowledge sources for word sense disambiguation, although the information sources she used were not independent, making it difficult to evaluate the contribution of each component. Our system makes use of strictly independent knowledge sources and is implemented within GATE whose plug-and-play architecture makes the evaluation of individual components more straightforward.

4.1 Stages in the Tagging Process

At the moment the sense tagger consists of six stages, the first two preprocess the text which is to be disambiguated with the remaining four carrying out the disambiguation.

1. The text is first processed by a named-entity identifier, which we developed as part of Sheffield's entry for MUC-6 [9]. This identifies certain forms of proper names in the text and classifies them as either place, person, organization or location. The tagger does not attempt to disambiguate any words which are identified as part of a named-entity.
2. The remaining text is stemmed, leaving only morphological roots, and split into sentences. Then words belonging to a list of stop words are removed.[3] The words which have not been identified as part of a named entity or removed because it is a stop word are considered by the system to be *ambiguous words* and those are the words which are disambiguated.
   For each of the ambiguous words, its set of possible senses is extracted from LDOCE and stored. Each sense in LDOCE contains a short textual definition which, when extracted from the dictionary, is processed to remove stop words and stem the remaining words.
3. The text is tagged using the Brill tagger [3] and a translation is carried out using a manually defined mapping from the syntactic tags assigned by Brill onto the simpler part-of-speech categories associated with LDOCE senses.[4] We then remove from consideration any of the senses whose part-of-speech is not consistent with the one assigned by the tagger, if none of the senses are consistent with the part-of-speech we assume the tagger has made an error and leave the set of senses for that word unaltered.
4. Our first partial tagger is based on the simulated annealing approach of Cowie et. al. [5] which disambiguates a sentence at a time. A random configuration of senses is chosen such that one sense is assigned to each ambiguous word in the sentence. A score is given to this configuration based on the number of content words which are shared between the textual definition in the senses. Other, random, configurations are then generated and the simulated annealing algorithm is used to optimise over them. When this process is complete the algorithm returns a configuration which assigns the optimal configuration of senses based on the overlap of words in the definition text.
   This process identifies a single candidate LDOCE sense for each ambiguous word.
5. The text is then run through a module which optimises the overlap of domain codes for the senses of nouns in each paragraph of the text. The optimisation algorithm used is similar to simulated annealing (see stage 4), although we have made some modifications to make the approch more suitable to domain code optimisation.
6. The final stage is to combine the results of the preceding processes. This is done using a very simple mechanism which we plan to replace with an optimisation algorithm. We assign a score to each of the senses of the ambiguous words. These scores are initialised to `0` and `+1` is added to a sense's score for each of the simulated annealing or pragmatic code modules which select that sense. The sense with the highest score is chosen as the tag for each ambiguous word. If there is a tie (two senses with the same score, which will happen if the two partial taggers disagree) it is broken by choosing the first sense, as listed in the dictionary. This is a sensible tie-breaker since the senses are roughly ordered by frequency of occurrence in text.[5] After this process is completed every ambiguous word has exactly one sense from LDOCE associated with it, this sense is the tag which our system has assigned to that word.

4.2 Evaluation

We have conducted some preliminary testing of our tagger: our tests were run on 14 hand-disambiguated sentences from the Wall Street Journal, amounting to a 314 word corpus. This is the same evaluation corpus which was used by Cowie et. al. [5]. We found that, of the tokens with more than 1 homograph in LDOCE, 88% were assigned the correct homograph and 60% the correct sense using our tagger. This equates to 93% correct homograph and 76% correct sense assignment over all words in the texts.

These figures should be compared to the 72% correct homograph assignment and 47% correct sense assignment reported by Cowie et. al. [5] using simulated annealing alone on the same test set. It should be noted that the granularity of sense distinctions at the LDOCE homograph level (eg. "bank" as 'edge of river' or 'financial institution') is comparable to the distinctions made by small-scale WSD algorithms (eg. [10], [18] and [25]) and our tagger is a true language engineering application operating on free text and using sense tags from a standard NLP lexicon.

One of the reasons for using multiple knowledge sources was to allow us to analyse the contribution made by each of the sources. This sort of information could be used in the design

[3] In our system a stop word is defined to be any word which is not a noun, verb, adjective or adverb. Prepositions are included in the list of stop words and are not disambiguated.

[4] The Brill tagger uses the tag set from the Penn Tree Bank which contains 48 tags, LDOCE uses a set of 17, more general, tags.

[5] We are using the 1st Edition of LDOCE in which the publishers make no claim that the senses are ordered by frequency of occurrence in text (although they do in later editions). However, [13] has found evidence that there is a correspondence between the order in which sense are listed and the frequency of occurrence.

of future semantic taggers, if, for example, we found that one of the modules was not performing much disambiguation we would not make further use of it. We analysed eight different configurations of the tagger, using neither of the partial taggers **(1)**(equivalent to choosing the first sense), using only the simulated annealing tagger **(2)**, only the pragmatic codes tagger **(3)** and using both taggers **(2)+(3)**. Each of these were compared with and without the filtering by part of speech codes and measured to both the sense and homograph level. The results are shown in table 1.

As we can see from this table the addition of partial taggers improves the results, although, the addition of more than one does not seem to alter the performance any more than a single tagger. There are two possible reasons for this, firstly, that each of the partial taggers are disambiguating the same words and so there is no advantage in using both and, secondly, that the taggers are conflicting with each others results – this could be solved by making use of a more intelligent method for combining their results. Further analysis is required before we can discover which is the case. In all cases the addition of the part of speech filtering improves the performance of the tagger.

Our evaluation is unsatisfactory due to the small size of the test data, although we hope it is at least indicative of the sorts of results we might find on a larger test corpus. However such corpora are relatively rare in NLP and we are currently exploring ways of producing sense tagged corpora in an automatic manner. We hope to analyse our tagger on a larger corpus and to explore more fully the relationship between the partial taggers in the future.

5 Lessons Learned

Our experience of implementing a novel system within GATE has given us insights into the usefulness of the architecture which we could not have gained otherwise, the overall result of the experience was very encouraging.

In the TIPSTER annotation scheme, GATE provides a well-defined standard for data-transfer between modules. This standard approach allows for the rapid re-use of existing modules and reduced the need to provide data-transfer routes between modules. Almost the entire preprocessing of the text was carried out using modules which had already been implemented within GATE: the tokeniser, sentence splitter, Brill part-of-speech tagger and the modules which made up the Named Entity identifier. This meant that we could quickly implement the modules which carried out the disambiguation, and those were the modules in which we were most interested. The implementation was further speeded up by the use of results viewers which allowed us to examine the annotations in the TIPSTER DataBase after a module had been run, allowing us to discover bugs far more quickly than would have been possible in a system which is not as explicitly modular as GATE. One aspect of sense tagging in which we are interested is the effect of including and excluding different modules, and this could be easily carried out using GGI.

One particular limitation of the current GATE implementation became apparent during this work, viz. the necessity of cascading module reset in the presence of non-monotonic database updates. For example, the POS filter modules remove some of the sense definitions associated with words by the lexical preprocessing stages. When reseting these modules it is therefore necessary to reset the preprocessor stage in order that the database is returned to a consistent state (this is done automatically by GATE, which identifies cases where modules alter previously existing annotations by examination of the pre-/post-conditions of the module supplied by the developer as configuration information prior to loading). This leads to redundant processing, and in the case of slow modules (like our LDOCE lookup module) this can be an appreciable brake on the development cycle. The planned solution is to change the implementation of the reset function. Currently this simply deletes the database objects created by a module. Given a database implementation that supports transactions we can use timestamp and rollback for a more intelligent reset, and avoid the redundant processing caused by reset cascading.

An additional, lesser problem, is the complexity of the generation algorithms for the task graphs, and the difficulty of managing these graphs as the number of modules in the system grows. The graphs currently make two main contributions to the system: they give a graphical representation of control flow, and allow the user to manipulate execution of modules; they give a graphical entry point to results visualisation. These benefits will have to be balanced against their disadvantages in future versions of the system. Another problem may arise when the architecture includes facilities for distributed processing [28, 29], as it is not obvious how the linear model currently embodied in the graphs could be extended to support non-linear control structures.

6 Conclusion

In this paper we have described two systems and explained their usefulness within the field of language engineering, although neither system has been fully developed. GATE version 1 goes a long way to meeting its design goals (noted in section 2). Our experience during the sense tagging project has proved its usefulness for the language engineer. Future versions will address the problems we uncovered above. Our sense tagger has a high rate of success when compared to other systems however, we wish to analyse its results further in order to gain more insights to the task. We believe GATE is the correct architecture in which to carry this out.

Acknowledgements

This work has been supported by the UK's EPSRC and the European Commission Language Engineering programme under grants GR/K25267 (GATE), LE1-2238 (AVENTINUS) and LE1-2110 (ECRAN).

We are also grateful for the comments made by the two anonymous reviewers of this paper.

REFERENCES

[1] J.W. Amtrup, 'ICE – INTARC Communication Environment User Guide and Reference Manual Version 1.4', Technical report, University of Hamburg, (1995).

[2] B. Boguraev, R. Garigliano, and J. Tait, 'Editorial.', *Natural Language Engineering.*, **1, Part 1.**, (1995.).

		(1)	(2)	(3)	(2)+(3)
POS filter off	**Homograph**	62%	72%	73%	74%
	Sense	40%	47%	48%	48%
POS filter on	**Homograph**	80%	86%	86%	88%
	Sense	54%	57%	58%	60%

Table 1. Results from different Configurations of the Tagger

[3] E. Brill, 'A simple rule-based part of speech tagger', in *Proceedings of the DARPA Speech and Natural Language Workshop*, Harriman, NY, (1992).

[4] R. Bruce and L. Guthrie, 'Genus disambiguation: A study in weighted preference', in *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, pp. 1187–1191, Nantes, France, (1992).

[5] J. Cowie, L. Guthrie, and J. Guthrie, 'Lexical disambiguation using simulated annealing', in *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, pp. 359–365, Nantes, France, (1992).

[6] H. Cunningham, 'Support Software for Language Engineering Research', Technical Report 94/05, Centre for Computational Linguistics, UMIST, Manchester, (1994).

[7] H. Cunningham, M. Freeman, and W.J. Black, 'Software Reuse, Object-Oriented Frameworks and Natural Language Processing', in *Proceedings of the Conference on New Methods in Natural Language Processing (NeMLaP-1)*, Manchester, (1994).

[8] H. Cunninham, K. Humphreys, R. Gaizauskas, and Y. Wilks, 'Software Infrastructure for Natural Language Processing', in *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, (March 1997). Available as `http://xxx.lanl.gov/ps/9702005`.

[9] R.G. Gaizauskas, H. Cunningham, Y. Wilks, P. Rodgers, and K. Humphreys, 'GATE – an Environment to Support Research and Development in Natural Language Engineering', in *Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-96)*, Toulouse, France, (October 1996).

[10] W. A. Gale, Church K. W., and Yarowsky D., 'One sense per discourse', in *Proceedings of the DARPA Speech and Natural Language Workshop*, Harriman, New York, (February 1992).

[11] G. Gazdar, 'Paradigm merger in natural language processing', in *Computing Tomorrow: Future Research Directions in Computer Science*, eds., R. Milner and I. Wand, 88–109, Cambridge University Press, (1996).

[12] R. Grishman, 'TIPSTER Architecture Design Document Version 2.3', Technical report, DARPA, (1997). Available at `http://www.tipster.org/`.

[13] C-M Guo, 'Constructing a Machine Tractable Dictionary from Longman Dictionary of Contemporary English', Technical Report MCCS-89-156, Computing Research Laboratory, New Mexico State University, (1989).

[14] K. Mahesh, S. Nirenburg, S. Beale, E. Viegas, V. Raskin, and B. Onyshkeyvych, 'Word sense disambiguation: Why have statistics when we have these numbers', in *Proceedings of the 7th International Conference on Theoretical and Methodological Issues in Machine Translation*, pp. 151–159, (July 1997).

[15] S. McRoy, 'Using multiple knowledge sources for word sense disambiguation', *Computational Linguistics*, **18**(1), 1–30, (1992).

[16] T. Pedersen and R. Bruce, 'Distinguishing word senses in untagged text', in *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, Providence, RI, (August 1997).

[17] P. Procter, *Longman Dictionary of Contemporary English*, Longman Group, Essex, England, 1978.

[18] H. Schütze, 'Dimensions of meaning', in *Proceedings of Supercomputing '92*, pp. 787–796, Minneapolis, MN, (1992).

[19] M. Shaw and D. Garlan, *Software Architecture*, Prentice Hall, New York, 1996.

[20] N. K. Simkins, 'An Open Architecture for Language Engineering', in *First CEC Language Engineering Convention, Paris*, (1994).

[21] H. Thompson and D. McKelvie, 'A Software Architecture for Simple, Efficient SGML Applications', in *Proceedings of SGML Europe '96, Munich*, (1996).

[22] Y. Wilks, 'Natural Language Processing, guest editorial', *Communications of the ACM*, **Volume 39**(Number 1), (January 1996).

[23] Y. Wilks, L. Guthrie, and B. Slator, *Electric Words: Lexicons, Dictionaries and Meanings*, MIT Press, Cambridge, MA, 1995.

[24] Y. Wilks and M. Stevenson, 'Sense tagging: Semantic tagging with a lexicon', in *Proceedings of the SIGLEX Workshop "Tagging Text with Lexical Semantics: What, why and how?"*, pp. 47–51. ANLP, (1997). Available as `http://xxx.lanl.gov/ps/cmp-lg/9705016`.

[25] D. Yarowsky, 'Word-sense disambiguation using statistical models of Roget's categories trained on large corpora', in *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, (1992).

[26] D. Yarowsky, 'One sense per collocation', in *Proceedings ARPA Human Language Technology Workshop*, pp. 266–271, (1993).

[27] D. Yarowsky, 'Unsupervised word-sense disambiguation rivaling supervised methods', in *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL '95)*, pp. 189–196, Cambridge, MA, (1995).

[28] R. Zajac, 'An Open Distributed Architecture for Reuse and Integration of Heterogenous NLP Components', in *Proceedings of the 5th conference on Applied Natural Language Processing (ANLP-97)*, (1997).

[29] R. Zajac, V. Mahesh, H. Pfeiffer, and M. Casper, 'The Corelli Document Processing architecture.', Technical report, Computing Research Lab, New Mexico State University, (1997.).