In Proceedings of the AISB'99 Workshop on Reference Architectures and Data Standards for NLP, Edinburgh, April, 1999.

Experience with a Language Engineering Architecture: Three Years of GATE

Hamish Cunningham; Robert Gaizauskas; Kevin Humphreys; Yorick Wilks
Natural Language Processing group,
Institute for Language, Speech and Hearing, and
Department of Computer Science,
University of Sheffield.
{hamish,robertg,kwh,yorick}@dcs.shef.ac.uk.

Abstract

GATE, the General Architecture for Text Engineering, aims to provide a software infrastructure for researchers and developers working in the area of natural language processing. A version of GATE has now been widely available for three years. In this paper we review the objectives which motivated the creation of GATE and the functionality and design of the current system. We discuss the strengths and weaknesses of the current system, identify areas for improvement and present plans for implementing these improvements.

Introduction

We think that if you're researching human language processing you should probably not be writing code to:

- store data on disk;
- display data;
- load processor modules and data stores into processes;
- initiate and administer processes;
- divide computation between client and server;
- pass data between processes and machines.

A *Domain-Specific Software Architecture* (DSSA) for language processing should do all this for you. You will have to parameterise it, and sometimes deployment of your work into applications software will require some low-level fiddling for optimisation purposes, but in the main these activities should be carried out by infrastructure for the language sciences, not by each researcher in the field.

We can go further and say that you shouldn't have to reinvent components and resources outside of your specialism if there is already something that could do the job. A statistician doesn't need to know the details of the IEEE Floating Point computation standard; a discourse processing specialist doesn't need to understand all the ins and outs of part-of-speech tagging (or worse still how to install a particular POS tagger on a particular machine).

If you're a professional mathematician, you probably regard a tool like SPSS or Mathematica as necessary infrastructure for your work. If you're a computational linguist or a language engineer, the chances are that large parts of your work have no such infrastructural support. Where there is infrastructure, it tends to be specific to restricted areas. GATE, a General Architecture for Text Engineering (Cunningham et al., 1997), represents an attempt to fill this gap, and is a DSSA for language processing R&D.

We now have three years of experience with GATE, work on which began in 1995, with a first widespread release late in 1996. The system is currently at a pivotal point in its development, with a new version on the drawing board, so it seems appropriate to cast a critical eye over progress so far, and then to present our plans for the future.

We begin in section 1 with context: what is an architecture and why might we want one? Section 2 describes GATE in its current form. Section 3 discusses the strengths and weaknesses of the system, and draws out some pointers for improvement to the system. Section 4 presents our plans for implementing these improvements, and the conclusion gives final remarks.

To view this text on-line, see
http://www.dcs.shef.ac.uk/ hamish/GateAisb99.html.

1 Infrastructure for Language Processing R&D

What does infrastructure mean for Natural Language Processing (NLP)? What sorts of tasks should be delegated to

a general tool, and which should be left to individual projects? The position we took in designing GATE is to focus on the *common elements of NLP systems*.

There are many useful tools around for performing specific tasks such as developing feature structure grammars for evaluation under unification, or collecting statistical measures across corpora. To varying extents, they entail the adoption of particular theories. The only common factor of NLP systems, alas, seems to be that they very often create information about text. Developers of such systems create modules and data resources that handle text, and they store this data, exchange it between various modules, compare results of test runs, and generally spend inordinate amounts of time pouring over samples of it when they really should be enjoying a slurp of something relaxing instead.

The types of data structure typically involved are large and complex, and without good tools to manage and allow succinct viewing of the data we work below our potential. At this stage in the progress of our field, no one should really have to write a tree viewing program for the output of a syntax analyser, for example, or even have to do significant work to get an existing viewing tool to process their data.

In addition, many common language processing tasks have been solved to an acceptable degree by previous work and should be reused. Instead of writing a new part of speech tagger, or sentence splitter, or list of common nominal compounds, we should have available a store of reusable tools and data that can be plugged into our new systems with minimal effort. Such reuse is much less common than it should be, often because of installation and integration problems that have to be solved afresh in each case (Cunningham et al., 1994).

In sum, we defined our infrastructure as an architecture and development environment, where an architecture is a macro-level organisational pattern for the components and data resources that make up a language processing system; a development environment adds graphical tools to access the services provided by the architecture.

2 GATE

GATE version 1.n does three things:

- manages textual data storage and exchange;
- supports visual assembly and execution of modular NLP systems plus visualisation of data structures associated with text;
- provides plug-in modularity of text processing components.

The architecture does this using three subsystems:

- GDM, the GATE Document Manager;
- GGI, the GATE Graphical Interface;
- CREOLE, a Collection of REusable Objects for Language Engineering.

GDM manages the information about texts produced and consumed by NLP processes; GGI provides visual access to this data and manages control flow; CREOLE is the set of resources so far integrated. Developers working with GATE begin with a subset of CREOLE that does some basic tasks, perhaps tokenisation, sentence and paragraph identification and part-of-speech tagging. They then add or modify modules for their specific tasks. They use a single API for accessing the data and for storing their data back into the central database. With a few lines of configuration information they allow the system to display their data in friendly graphical form, including tree diagrams where appropriate. The system takes care of data storage and module loading, and can be used to deliver embeddable subsystems by stripping the graphical interface. It supports modules in any language including Prolog, Lisp, Perl, Java, C++ and Tcl.

3 Strengths and Weaknesses

GATE has proved successful in a number of contexts, with users reporting a variety of work with the system, for example:

- Teaching undergraduates and postgraduates. Our colleagues at UMIST and the Universities of Edinburgh, and Sussex have reported using the system for teaching, as have the Universities of Stuttgart and Saarburcken.
- Information Extraction in English, Swedish, French, Spanish and Greek. Our colleagues in Fribourg University collaborated with us on a French IE system; both ILSP and NKSR Demokritus in Athens are developing a Greek IE system; the University of Gothenburg has a Swedish system; the University of Catalonia in Barcelona are working on Spanish.
- Integrating information extraction with Information Retrieval. The Naval Office of R&D (NRaD) in San Diego is using GATE for research on text summarisation and IE/IR integration.
- Integrating a national collection of NLP tools for Swedish. See http://www.sics.se/humle/projects/svensk/
- ESTEAM Inc., of Gothenburg and Athens are using the system for adding name recognition to their MT systems (for 26 language pairs) to improve performance on unknowns.
- The Speech and Hearing group at Sheffield are modelling out-of-vocabulary language using VIE and GATE (Gotoh et al., 1998).

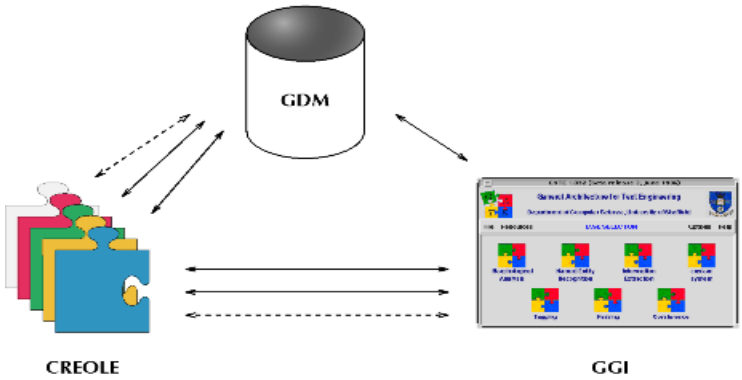

Figure 1: Gate Architecture

- Numerous postgraduates in locations as diverse as Israel, Copenhagen and Surrey are using the system to avoid having to write simple things like sentence splitters from scratch, and to enable visualisation and management of data.

Abstracting from their experiences and that of users at Sheffield, GATE's strengths can be summarised as:

- facilitating reuse of NLP components by reducing the overheads of integration, documentation and data visualisation;
- facilitating multi-site collaboration on IE research by providing a modular base-line system (VIE) with which others can experiment;
- facilitating comparative evaluation of different methods by making it easy to interchange modules;
- facilitating task-based evaluation, both of "internal" components such as taggers and parsers, and of whole systems, e.g. by using materials from the ARPA MUC programme (Grishman and Sundheim, 1996) (whose scoring software is available in GATE, as is the Parseval tree scoring tool (Harrison et al., 1991), and a generic annotation scoring tool);
- contributing to the portability of NLP systems across problem domains by providing a markup tool for generating training data for example-based learning (it can also take input from the Alembic tool (Day et al., 1997) for this purpose, using Edinburgh's SGML processing library (McKelvie et al., 1997)).

There several weaknesses in the system, and some areas that are underdeveloped or lacking polish. In rough order of severity:

1. Version 1 is biased towards algorithmic components for language processing, and neglects resource components.
2. Version 1 is biased towards text analysis components, and neglects text generation components.
3. The visual interface is complex and somewhat non-standard.
4. Installing and supporting the system is a skilled job, and it runs better on some platforms than on others (UNIX vs. Windows).
5. Sharing of modules depends on sharing of annotation definitions (but isomorphic transformations are relatively easy to implement).
6. It only caters for textual documents, not for multimedia documents.
7. It only supports 8-bit character sets.

Points 1. and 2. compromise the generality of the system, and have limited take-up, as well as the number of CREOLE modules integrated with the system. For modules like taggers, parsers, discourse analysers (i.e. just about anything that performs an analysis task) the GATE integration model provides a convenient and powerful abstraction layer based on storing information in association with the text under analysis. For resources like lexicons or corpora, no such layer exists. Similarly, for modules that do generation-side tasks, since there is no text under analysis, the utility of a text-based model is limited.

4 Version 2

Our main aim is to extend CREOLE to cover as many core areas of language engineering R&D as possible. In support of this aim, GDM and GGI will be developed in a number of significant ways; for the purposes of this abstract we will discuss only one, viz. the extension of the currently process-oriented resources set to include predominantly data resources such as lexicons, ontologies, thesauri and corpora. Much progress has been made over the last decade in provision of large-scale resources of this type (Wilks et al., 1996), but despite various standards initiatives, there are still barriers to data resource reuse:

- each resource has its own representation syntax and corresponding access mode: relational databases in the case of CELEX, a custom C API or Prolog representation in Wordnet, SGML in the case of the British National Corpus;
- resources must be installed locally to be usable (and precisely how this happens, what operating systems are supported etc., varies from site to site).

The consequences of the first bullet are that although linguistic resources normally share some structure (e.g. at the most obvious level, lexicons are organised around words and word strings) this commonality is wasted when using a new resource, since the developer has to learn everything afresh, and work which seeks to investigate or exploit commonalities between resources (e.g. to link several lexicons to an ontology) has first to build a layer of access routines on top of each resource. So, for example, if we wish to do task-based evaluation of lexicons by measuring the relative performance of an information extraction system with different instantiations of the lexical resource, we might end up writing code to translate several different resources into SQL or SGML.

The consequence of the second bullet is that there is no way to "try before you buy": no way to examine a data resource for its suitability for your needs before licensing it. Correspondingly, there is no way for a resource provider to give limited access to their products for advertising purposes, nor gain revenue through piecemeal supply of sections of a resource. To address this problem we plan to develop a common model of the various resources types, implemented in Java, along with a distributed server for non-local access, and distribute the code required to map them into this model.

The common model of language data resources we propose is a set of inheritance hierarchies. At the top of the hierarchies are very general abstractions from resources (e.g. a thesaurus groups synonyms); at the leaves are data items specific to individual resources (e.g. WordNet synsets have glosses). Program access will be available at all levels, allowing the developer to select an appropriate level of commonality for each application. Note that, although an exciting element of the work is to provide algorithms dynamically to link common resources e.g. connecting EuroWordNet to LDOCE, this proposal is not to develop new resources, but simply to improve access to existing ones. Notice, also, that it is a proposal about language data quite separate from, though compatible with, the lexical data compression ideas in recent DATR work (Evans and Gazdar, 1996). Finally, this is NOT in any way a new standards initiative, but a way to build on previous initiatives. The issues of standards is a vexed one: experience with repositories of lexical materials (e.g. the CRL Consortium for Lexical Research 1989-93) suggests that if resources must have standardised formats, they are not deposited or used. Our proposal acknowledges the *de facto* diversity of lexical resource format, but attempts to render these resources more usable for language engineering. More details of the approach may be found in Cunningham et al. (1998).

Conclusion

The GATE system is used in several EC Fourth Framework projects and is installed at over 200 sites worldwide. It was adopted by the US TIPSTER programme as part of their final Architecture Capabilities Platform, and is available free of charge on licence from Sheffield.

Based on the collective experiences of a sizeable user base across the EU and elsewhere, the system can claim to be a viable NLP DSSA for certain sections of the field. Given further development, we hope that it can take on this role for a wider variety of tasks.

Acknowledgements

This work was supported by EPSRC grant GR/K25267, NATO science grant CRG 960752, two ARPA contracts under the TIPSTER programme and by a contract with the Max Planck Institute in Nijmegen.

References

H. Cunningham, M. Freeman, and W.J. Black. Software Reuse, Object-Oriented Frameworks and Natural Language Processing. In *New Methods in Language Processing (NeMLaP-1), September 1994*, Manchester, 1994. Published in 1997 by UCL Press.

H. Cunningham, 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)*, pages 237–244, 1997. Available as `http://xxx.lanl.gov/ps/9702005`.

H. Cunningham, W. Peters, C. McCauley, K. Bontcheva, and Y. Wilks. A Level Playing Field for Language Resource Evaluation. In *Workshop on Distributing and Accessing Lexical Resources at Conference on Language Resources Evaluation, Granada, Spain*, 1998.

D. Day, J. Aberdeen, L. Hirschman, R. Kozierok, P. Robinson, and M. Vilain. Mixed-Initiative Development of Language Processing Systems. In *Proceedings of the 5th Conference on Applied NLP Systems (ANLP-97)*, 1997.

R. Evans and G. Gazdar. Datr: A language for lexical knowledge representation. *Compuational Linguistics*, 22(1), 1996.

Yoshihiko Gotoh, Steve Renals, Rob Gaizauskas, Gethin Williams, and Hamish Cunningham. Named entity

tagged language models for LVCSR. Technical Report CS-98-05, Department of Computer Science, University of Sheffield, 1998.

R. Grishman and B. Sundheim. Message understanding conference - 6: A brief history. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 466–471, Copenhagen, 1996.

P. Harrison, S. Abney, E. Black, D. Flickinger, C. Gdaniec, R. Grishman, D. Hindle, R. Ingria, M. Marcus, B. Santorini, and T. Strzalkowski. Evaluating syntax performance of parser/grammars of english. In *Proceedings of the Workshop On Evaluating Natural Language Processing Systems*. Association For Computational Linguistics, 1991.

D. McKelvie, C. Brew, and H. Thompson. Using SGML as a Basis for Data-Intensive NLP. In *Proceedings of the fifth Conference on Applied Natural Language Processing (ANLP-97)*, 1997.

Y. Wilks, L. Guthrie, and B. Slator. *Electric Words*. MIT Press, Cambridge, MA, 1996.