GATE – a General Architecture for Text Engineering

Hamish Cunningham
Dept. Computer Science
University of Sheffield
211 Portobello St.
Sheffield S10 4DP
hamish@dcs.shef.ac.uk

Kevin Humphreys
Dept. Computer Science
University of Sheffield
211 Portobello St.
Sheffield S10 4DP
kwh@dcs.shef.ac.uk

Robert Gaizauskas
Dept. Computer Science
University of Sheffield
211 Portobello St.
Sheffield S10 4DP
robertg@dcs.shef.ac.uk

Yorick Wilks
Dept. Computer Science
University of Sheffield
211 Portobello St.
Sheffield S10 4DP
yorick@dcs.shef.ac.uk

For a variety of reasons NLP has recently spawned a related engineering discipline called *language engineering* (LE), whose orientation is towards the application of NLP techniques to solving large-scale, real-world language processing problems in a robust and predictable way. Aside from the host of fundamental theoretical problems that remain to be answered in NLP, language engineering faces a variety of problems of its own. First, there is no theory of language which is universally accepted, and no computational model of even a part of the process of language understanding which stands uncontested. Second, building intelligent application systems, systems which model or reproduce enough human language processing capability to be useful, is a large-scale engineering effort which, given political and economic realities, must rely on the efforts of many small groups of researchers, spatially and temporally distributed. The first point means that any attempt to push researchers into a theoretical or representational straight-jacket is premature, unhealthy and doomed to failure. The second means that no research team alone is likely to have the resources to build from scratch an entire state-of-the-art LE application system.

Given this state of affairs, what is the best practical support that can be given to advance the field? Clearly, the pressure to build on the efforts of others demands that LE tools or component technologies be readily available for experimentation and reuse. But the pressure towards theoretical diversity means that there is no point attempting to gain agreement, in the short term, on what set of component technologies should be developed or on the informational content or syntax of representations that these components should require or produce.

Our response has been to design and implement a software environment called GATE (Cunninham et al., 1997), which we will demonstrate at ANLP. GATE attempts to meet the following objectives:

1. support information interchange between LE modules at the highest common level possible without prescribing theoretical approach;
2. support the integration of modules written in any source language on any common platform;
3. support the evaluation and refinement of LE component modules, and of systems built from them, via a uniform, easy-to-use graphical interface which in addition offers facilities for visualising data and managing corpora.

Corresponding to the three key objectives identified above GATE comprises three principal elements: GDM, the GATE Document Manager, based on the TIPSTER document manager; CREOLE, a Collection of REusable Objects for Language Engineering: a set of LE modules integrated with the system; and GGI, the GATE Graphical Interface, a development tool for LE R&D, providing integrated access to the services of the other components and adding visualisation and debugging tools.

The GDM provides a central repository or server that stores all information an LE system generates about the texts it processes. All communication between the components of an LE system goes through GDM, which insulates these components from direct contact with each other and provides them with a uniform API for manipulating the data they produce and consume. The basic concepts of the data model underlying the GDM are those of the TIPSTER architecture, which is specified (Grishman, 1996).

All the real work of analysing texts in a GATE-based LE system is done by CREOLE modules or objects (we use the terms *module* and *object* rather loosely to mean interfaces to resources which may be predominantly algorithmic or predominantly data, or a mixture of both). Typically, a CREOLE object will be a wrapper around a pre-existing LE module or database – a tagger or parser, a lexicon or ngram index, for example. Alternatively, objects may be developed from scratch for the architecture – in either case the object provides a standardised API to the underlying resources which allows access via GGI and I/O via GDM. The CREOLE APIs may also be used for programming new objects.

When the user initiates a particular CREOLE object via GGI (or when a programmer does the same via the GATE API when building an LE application) the object is run, obtaining the information it needs (document source, annotations from other objects) via calls to the GDM API. Its results are then stored in the GDM database and become available for examination via GGI or to be the input to other CREOLE objects.

GDM imposes constraints on the I/O format of CREOLE objects, namely that all information must be associated with byte offsets and conform to the annotations model of the TIPSTER architecture. The principal overhead in integrating a module with GATE is making the components use byte offsets, if they do not already do so.

The GGI is a graphical tool that encapsulates the GDM and CREOLE resources in a fashion suitable for interactive building and testing of LE components and systems. The 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. The GGI 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 CREOLE developers reuse GATE data visualisation code with negligible overhead.

A central function of the GGI is to provide a graphical launchpad for the various LE subsystems available in GATE. To that end, the main panel of the GGI top-level display shows the particular tasks which may be performed by modules or systems within the GATE system (e.g. parsing). Having chosen a task, a window appears displaying a connected graph of the modules that need to be run to achieve the task. In this graph, the boxes denoting modules are active buttons: clicking on them will, if conditions are right, cause the module to be executed. The paths through the graph indicate the dependencies amongst the various modules making up this subsystem. At any point in time, the state of execution of the system, or, more accurately, the availability of data from various modules, is depicted through colour-coding of the module boxes. After execution, the results of completed modules are available for viewing by clicking again on the module box, and are displayed using an appropriate annotation viewer as described above. In addition, modules can be 'reset', i.e. their results removed from the GDM, to allow the user to pick another path through the graph, or re-execute having altered some tailorable data-resource (such as a grammar or lexicon) interpreted by the module at run-time. (Modules running as external executables might also be recompiled between runs.)

To illustrate the process of converting pre-existing LE systems into GATE-compatible CREOLE sets we use as an example the creation of VIE (Vanilla Information Extraction system) from LaSIE (Large-Scale Information Extraction system) (Gaizauskas et al., 1995), Sheffield's entry in the MUC-6 system evaluations. LaSIE module interfaces were not standardised when originally produced and its CREOLEization gives a good indication of the ease of integrating other LE tools into GATE. The work took around 2 person-months. The resulting system, VIE, is distributed with GATE.

References

Cunninham, H., K. Humphreys, R. Gaizauskas, and Y. Wilks. 1997. Software Infrastructure for Natural Language Processing. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, March.

Gaizauskas, R., T. Wakao, K Humphreys, H. Cunningham, and Y. Wilks. 1995. Description of the LaSIE system as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*. Morgan Kaufmann.

Grishman, R. 1996. TIPSTER Architecture Design Document Version 2.2. Technical report, DARPA. Available at `http://www.tipster.org/`.