Developing Reusable and Robust Language Processing Components for Information Systems using GATE

Kaling Bontcheva, Hamish Cunningham, Diana Maynard, Valentin Tablan, Horacio Saggion
Department of Computer Science,University of Sheffield
Regent Court, 211 Portobello Street, Sheffield, S1 4DP, UK
{kalina,hamish,diana,valyt,saggion}@dcs.shef.ac.uk

Abstract

In this paper we present GATE, an architecture and a graphical development environment which enables users to develop and deploy HLT applications in a robust fashion. GATE also provides reusable, extendable, and customisable language processing modules (e.g., part of speech tagger, named entity recognition grammars), which combined with the extensive document format support (e.g., XML, HTML), form a useful toolset for building HLT-augmented information systems.

1 Introduction

Information systems can benefit fully from Human Language Technology (HLT) only if the latter offers robust, scalable, and easy to integrate solutions. Producing such robust components requires attention to the engineering aspects of their construction. For instance, information systems often need to deal with large amounts of data, in multiple media, languages, formats, and locations (i.e., distributed over a network), so their language processing components need to be able to handle such data adequately. In addition, these components need to store the results so they could be easily retrieved and visualised later. For example, a named entity recognition module, working as part of a news Web portal, should be able to process quickly and with a minimum overhead the constant stream of news, which could come in XML, HTML, PDF, and other formats, and be located on several servers. Once the names are identified (e.g., persons, locations, dates), they need to be stored and indexed for later retrieval and visualisation, e.g., users could ask for sentences from today's news where President Bush and the Middle East are mentioned.

As argued in [3], all of these tasks can be addressed successfully by a framework for language processing software development and any HLT component built using this framework will benefit from these facilities, without having to re-implement them for each application, domain, or language. In this paper we will demonstrate how GATE version 2 supports the development and deployment of reusable and robust language processing components. GATE is available freely, as an open source system, under the GNU library licence from `http://gate.ac.uk` (including extensive documentation, research papers, tutorials and demos).

2 A framework for building robust tools and applications

GATE [4, 8] is an architecture, development environment, and framework for building systems and components that process human language. In this section we will discuss how it enables the low-cost creation of robust and reusable language processing components by providing the necessary engineering support for efficiency, data storage, format handling, retrieval, and visualisation.

Figure 1 shows a diagram of GATE's internal architecture and the way it addresses these issues. The core is the common data model of *annotation*, which associates linguistic and formatting data with document content. Annotations consist of a *type* and a set of *features* represented as attribute-value pairs. They are a powerful representation model, which is independent of any particular linguistic theory and can represent linguistic and document formatting data, regardless of their complexity. GATE's annotation model is a modified version of the TIPSTER format [6] which has been made largely compatible with the Atlas format [1].

GATE has an extendable set of document format handlers (e.g., XML, HTML, RTF, email), which translate the document content and the formatting information into GATE's shared data model.Handlers for new formats or media can easily be added by the user. For example, a new audio/video format handler can contain an OCR module

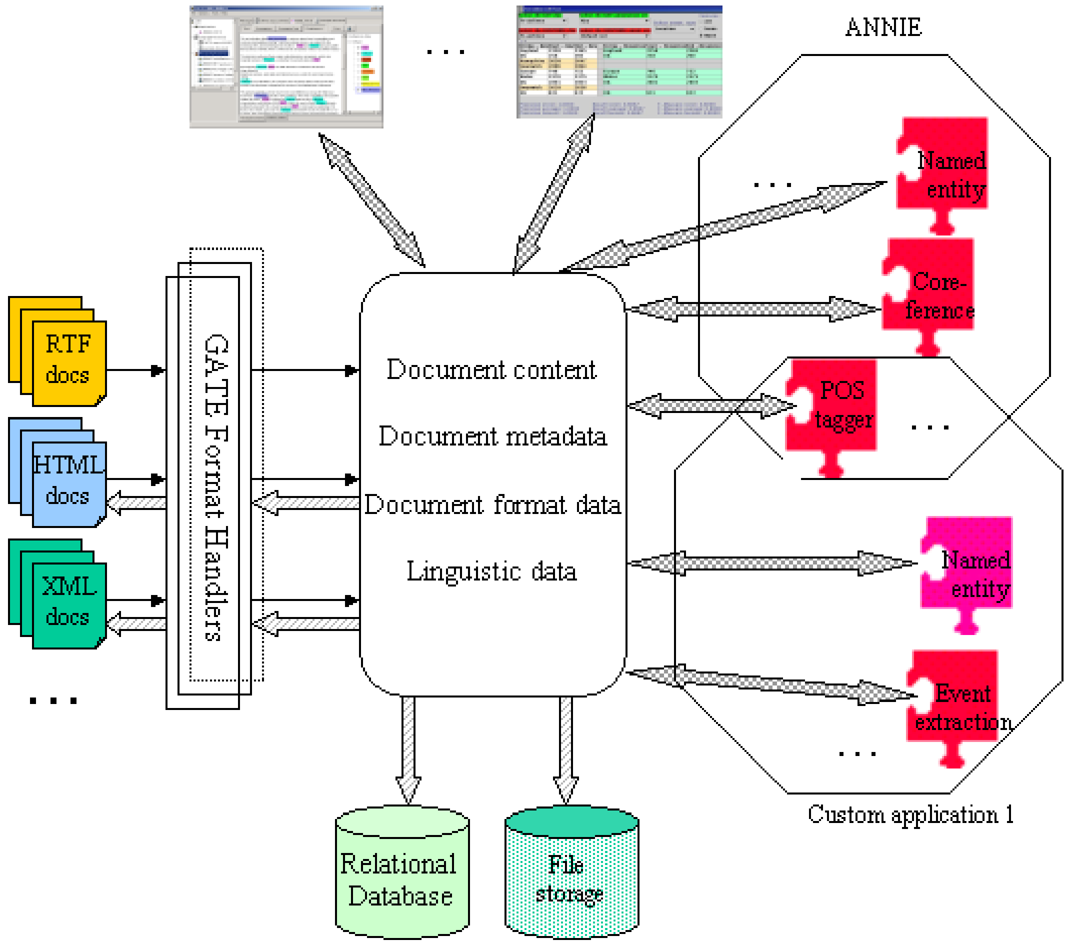

Figure 1. Data storage, format handling, modules, and applications in GATE

which is used to create the textual content of the document. Information about correspondence between text and media time and the URL of the multimedia source are stored in GATE's data model [2]. The automatic translation of the multimedia source into textual document content allows applications to reuse language processing modules, originally created for textual documents. Alternatively, these modules can be modified or new ones created, specifically targeted at processing OCR output and/or multimedia content.

The separation between format and media handling and the language processing components enables the latter to run on any document content, regardless of its origin, i.e., when support for a new format is added, the processing modules do not need to be changed. GATE also offers a comprehensive and module-independent multilingual support, which is based on Unicode [9].

The information in GATE's data model can be stored in three different ways, all of which are supported transparently to the processing modules (see Figure 1). It is down to the particular application to choose which persistence mechanism is best suited for its purpose. Again, no changes are required to any of the processing modules, because their input and output is abstracted away from the storage and retrieval technicalities. The three ways of storing the data are: *(i)* relational databases (Oracle, PostgreSQL); *(ii)* file-based storage, using Java's serialisation mechanism; *(iii)* as markup in XML format. Relational database storage allows the efficient execution of queries like 'Find all sentences in all documents, which mention President Bush and Israel'. Data stored in a database can also be accessed by non-GATE applications via SQL. Markup-based storage also allows non-GATE applications to access directly the language processing results, which are inserted as markup in the documents (e.g., <Person>Tony Blair</Person>).

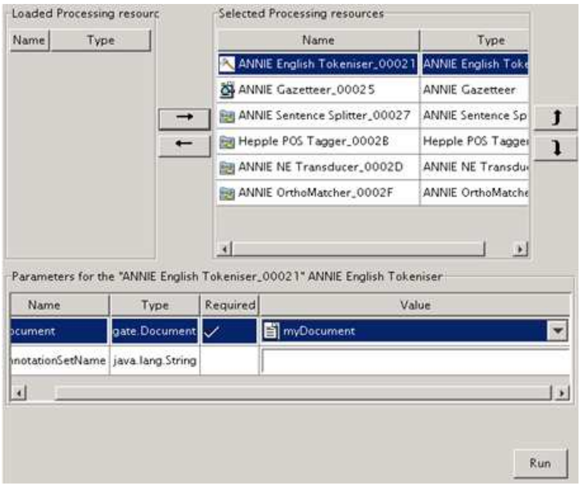

Figure 2. The application editor

3 Building HLT components and applications with GATE

As shown in Figure 1, GATE applications have a modular structure and consist of a number of language processing components (called Processing Resources or PRs). The same module can be included in more than one application (e.g., the POS tagger module), existing modules can be customised for new applications (e.g., the named entity module from ANNIE is frequently customised for new applications), and new application-specific modules can be created (e.g., the even extractio module in Figure 1).

In order to lower the application building overheads, GATE provides a number of useful and easy-to-customise components, grouped together to form ANNIE - A Nearly-New Information Extraction system[1]. These PRs eliminate the need for users to keep reimplementing frequently needed components and provide a good starting point for new applications. The majority of these PRs use GATE's finite state techniques to implement various tasks from tokenisation to semantic tagging and coreference. The emphasis is on efficiency, robustness, and low-overhead portability, rather than full parsing and deep semantic analysis.

The customisation of these modules and/or the creation of new ones typically requires knowledge of GATE's pattern action rule language called JAPE [5] and a basic understanding of the annotation data model. The visual development environment provides tools to support this process in a number of ways: data visualisation for debugging and measurement; component-based development in which alternative configurations can be compared straightforwardly; automated performance evaluation (for further details see [4]).

The JAPE language allows users to customise/create language processing components by creating and editing linguistic data (e.g., grammars, rule sets), while their efficient execution is automatically provided by GATE. Once familiar with GATE's data model, users do not find it difficult to write JAPE pattern-matching rules, because they are effectively regular expressions. An example rule below shows a typical JAPE rule. The left-hand side describes patterns of annotations that need to be matched, whereas the right-hand side specifies annotations to be created as a result:

```
Rule: Company1
Priority: 25
  (
    ({Token.orthography == upperInitial})+
    {Lookup.kind == companyDesignator}
  ):companyMatch
-->
  :companyMatch.NamedEntity =
    {kind = "company", rule = "Company1"}
```

The rule matches a pattern consisting of any kind of word (expressed as a Token annotation, created by a previous text tokenisation module), which starts with an upper-case letter, followed by a word which typically indicates companies, such as 'Ltd.' and 'GmbH' (the Lookup annotation). The entire match is then annotated with entity type "NamedEntity", and given a feature "kind" with value "company" and another feature "rule" with value "Company1". The "rule" feature is simply used for debugging purposes, so it is clear which particular rule has fired to create the annotation. For example, when this rule is applied to a document containing "Hewlett Packard Ltd.", the left-hand side will match, because there are two tokens starting with a capital letter, followed by the company designator 'Ltd.'. Due to space limitations, we will not provide further details on the ANNIE modules for tokenisation, gazetteer lookup, and part-of-speech tagging, which create the annotations which are used in the JAPE-based grammars, but see [8].

As shown in Figure 1, the processing components are structured to create a modular application. GATE's graphical environment makes this process easier by allowing users to construct the applications visually. The user chooses which PRs will be included (e.g. tokeniser, POS tagger), in what order they will be executed, and on which data. Figure 2 shows how the ANNIE application is built from the freely provided set of components. The execution order is manipulated by using the up and down arrow buttons. The execution parameters of each resource are also set there, e.g. a loaded document is given as a parameter

[1] A demonstration of how these components can be used to highlight information in Web pages is available at `http://gate.ac.uk/annie/index.jsp`.

to each PR. When the application is run, the modules are executed in the specified order on the given data.

Another substantial part of component and application development is debugging and performance measurement. GATE comes with a number of such tools which facilitate this process (see Figures 1 and 3). Since new components and applications might also require their own visualisation tool, GATE allows users to extend the current tool set, in a simple manner, similar to adding a new processing resource [5].

4 Applications

One of GATE's strengths is that it is flexible and robust enough to enable the development of a wide range of applications within its framework. In this section, we describe briefly some of the NLP applications we have developed using the GATE architecture. For an in-depth discussion of the effort involved in developing processing components in GATE see [7]. In all applications developed to this date, GATE has proved to reduce the development effort required, because the low-level technical tasks, such as data storage, are already catered for. Therefore, application development in most cases requires the implementation of new grammar rules, modifications to existing components, and/or creation of new ones. The effort on these tasks is also reduced due to the support of GATE's development environment.

Feedback from GATE users, who had to learn the system prior to developing applications with it, has shown that the substantial documentation, tutorials, and examples have helped them to complete their projects successfully. The usefulness of the provided set of customisable modules and the development environment have been pointed out as a major advantage. One of GATE's unique features, which is particularly relevant to information systems, is its use of relational databases for efficient storage and retrieval of language processing results.

4.1 MUMIS

The MUMIS (MUltiMedia Indexing and Searching environment) system uses Information Extraction (IE) components developed within GATE to produce formal annotations about essential events in football video programme material. This IE system comprises customised versions of the freely-available GATE modules and also a set of new IE modules such as event extraction. The effort involved in customising the existing modules was much lower than the effort that would have been needed for their development from scratch. The application also needs to handle a wide range of formats and media, including OCR content from the audio and video documents. Without GATE's comprehensive format support this alone would have required at least several person months to develop.

4.2 Summarisation

We have also adapted ANNIE to form an IE system (HaSIE), which aims at extracting relevant information from annual company reports about the companies' performance on Health and Safety issues. The extracted information allows the automated production of statistical metrics describing the level of compliance with Health and Safety recommendations and any relevant legislation that may be implemented. Although this application required substantial changes to the entity types recognised by the default ANNIE system, it was actually very simple to adapt the grammar to this new domain and application (the entire project from specification to completion required only 6 person months).

5 Conclusions

In this paper we have described GATE – an infrastructure which aims to assist the development of robust, scalable, and reusable HLT modules which can be integrated in information systems and other applications. The benefits of using such HLT infrastructure were discussed thoroughly and exemplified by our experience with working on a number of projects, including multimedia indexing and search among others. We have also just completed the provision of information retrieval support within GATE, which will allow the construction of information systems which have both IR and HLT capabilities.

Acknowledgments

Work on GATE has been supported by the Engineering and Physical Sciences Research Council (EPSRC) under grants GR/K25267 and GR/M31699, and by several smaller grants. The MUMIS project `http://mumis.vda.nl/` is funded by the EC's 5th Framework HLT programme under grant number IST-1999-10651. Project partners: Universities of Twente, Nijmegen and Sheffield, Esteam AB, VDA Ltd., DFKI.

References

[1] S. Bird, D. Day, J. Garofolo, J. Henderson, C. Laprun, and M. Liberman. ATLAS: A flexible and extensible architecture for linguistic annotation. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, Athens, 2000.

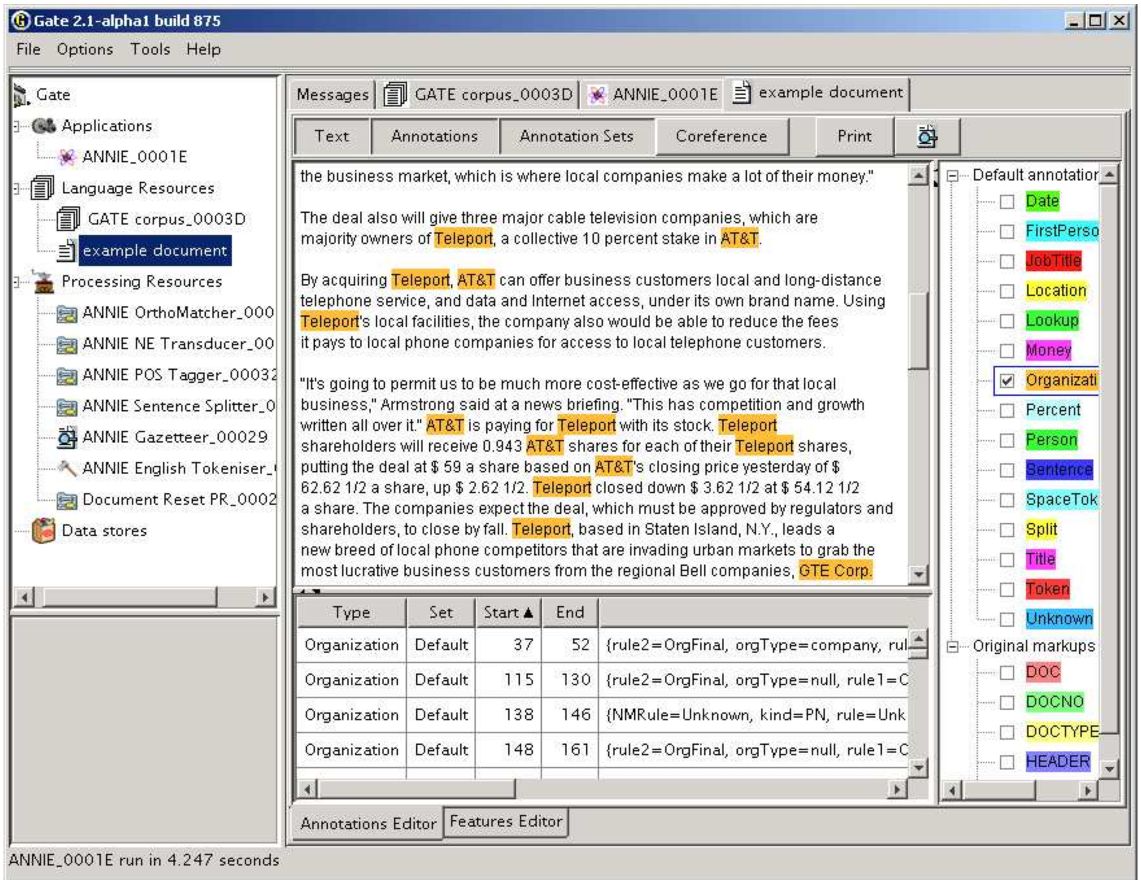

Figure 3. GATE's application development environment

[2] K. Bontcheva, H. Brugman, A. Russel, P. Wittenburg, and H. Cunningham. An Experiment in Unifying Audio-Visual and Textual Infrastructures for Language Processing R&D. In *Proceedings of the Workshop on Using Toolsets and Architectures To Build NLP Systems at COLING-2000*, Luxembourg, 2000. `http://gate.ac.uk/`.

[3] H. Cunningham, K. Bontcheva, V. Tablan, and Y. Wilks. Software Infrastructure for Language Resources: a Taxonomy of Previous Work and a Requirements Analysis. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC-2)*, Athens, 2000. `http://gate.ac.uk/`.

[4] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

[5] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, and C. Ursu. *The GATE User Guide*. `http://gate.ac.uk/`, 2002.

[6] R. Grishman. TIPSTER Architecture Design Document Version 2.3. Technical report, DARPA, 1997. `http://www.itl.nist.gov/div894/894.02/related_projects/tipster/`.

[7] D. Maynard, H. Cunningham, K. Bontcheva, O. Hamza, V. Tablan, and M. Dimitrov. Adapting an information extraction system to different domains, languages and applications. In *LREC Workshop on Customising Knowledge in NLP Applications*, 2002.

[8] D. Maynard, V. Tablan, H. Cunningham, C. Ursu, H. Saggion, K. Bontcheva, and Y. Wilks. Architectural elements of language engineering robustness. *Journal of Natural Language Engineering – Special Issue on Robust Methods in Analysis of Natural Language Data*, 2002. forthcoming.

[9] V. Tablan, C. Ursu, K. Bontcheva, H. Cunningham, D. Maynard, O. Hamza, T. McEnery, P. Baker, and M. Leisher. A unicode-based environment for creation and use of language resources. In *Proceedings of 3rd Language Resources and Evaluation Conference*, 2002. forthcoming.